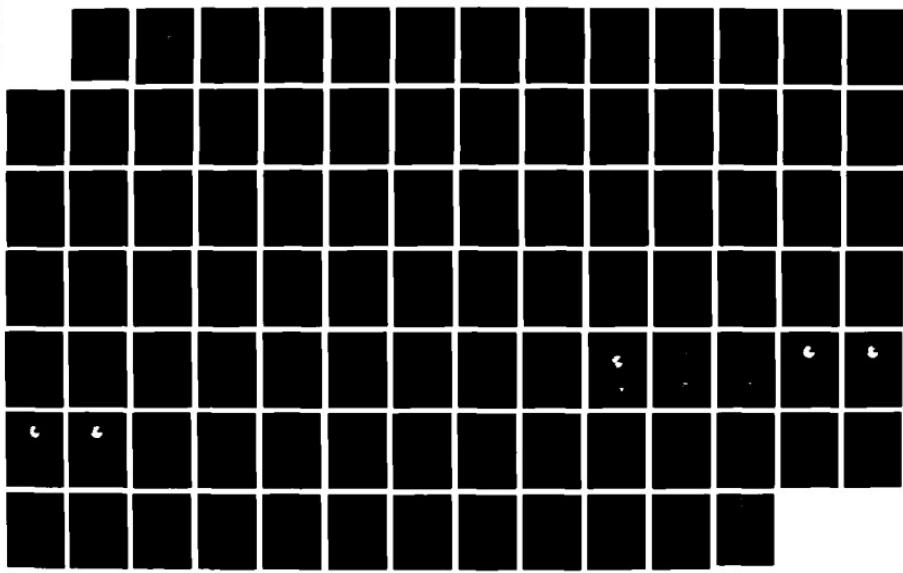
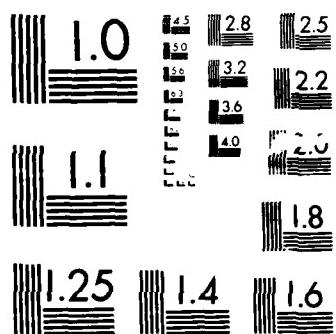


AB-R287 518 AN EXPERT SYSTEM IN C FOR COMPUTER-AIDED DIGITAL
CIRCUIT DESIGN(U) AIR FORCE INST OF TECH 171
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING
UNCLASSIFIED J S SANTOS JUN 89 AFIT/GCS/ENG/89J-1 F/G 20/3 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1962 A

AD-A207 518

(1)



AN EXPERT SYSTEM IN C FOR COMPUTER-AIDED
DIGITAL CIRCUIT DESIGN

THESIS

Jorge da Silva Santos
Captain, BRAZILIAN AIR FORCE

AFIT/GCS/ENG/89J-1

Approved for public release; distribution unlimited

189 5 05 040

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

ADA207518Form Approved
OMB No. 0704-0188**REPORT DOCUMENTATION PAGE**

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS							
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited							
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE									
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/ENG/89J-1		5. MONITORING ORGANIZATION REPORT NUMBER(S)							
6a. NAME OF PERFORMING ORGANIZATION School of Engineering	6b. OFFICE SYMBOL <i>(If applicable)</i> AFIT/ENA	7a. NAME OF MONITORING ORGANIZATION							
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433		7b. ADDRESS (City, State, and ZIP Code)							
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b. OFFICE SYMBOL <i>(If applicable)</i>	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER							
8c. ADDRESS (City, State, and ZIP Code)		10. SOURCE OF FUNDING NUMBERS							
		PROGRAM ELEMENT NO.	PROJECT NO.						
		TASK NO.	WORK UNIT ACCESSION NO.						
11. TITLE <i>(Include Security Classification)</i> An Expert System in C for Computer-Aided Digital Circuit Design UNCLASSIFIED									
12. PERSONAL AUTHOR(S) Jorge da Silva Santos, Captain, Brazilian Air Force									
13a. TYPE OF REPORT MS Thesis	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT <i>(Year, Month, Day)</i> 1989 June	15. PAGE COUNT 91						
16. SUPPLEMENTARY NOTATION									
17. COSATI CODES <table border="1"> <tr> <th>FIELD</th> <th>GROUP</th> <th>SUB-GROUP</th> </tr> <tr> <td>12</td> <td>05</td> <td></td> </tr> </table>		FIELD	GROUP	SUB-GROUP	12	05		18. SUBJECT TERMS <i>(Continue on reverse if necessary and identify by block number)</i> Expert System, C, Digital Circuit Computer Aided Design	
FIELD	GROUP	SUB-GROUP							
12	05								
19. ABSTRACT <i>(Continue on reverse if necessary and identify by block number)</i> Thesis Advisor: David W. Fautheree, Captain, USAF Instructor of Electrical Engineering and Computer Science									
<p>Abstract: This thesis effort documents the design, development, implementation, and test of an expert system which decomposes digital circuits into subproblems in order to detect wiring errors, which consist of improperly connected gates, missing connections, and violation of fanout or race conditions. Information needed to connect chips together is viewed as knowledge base information for the expert system. Information such type as number of pins, value of each pin (input, output, power, ground, clock), cont on reverse </p>									
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED							
22a. NAME OF RESPONSIBLE INDIVIDUAL David W. Fautheree, Captain, USAF		22b. TELEPHONE <i>(Include Area Code)</i> 513-255-3576	22c. OFFICE SYMBOL AFIT/ENG						

UNCLASSIFIED

Box 19 continued:

fanout for a particular type of chip are retrieved from a central database where they are represented.) The approach to this effort includes a examination of existing expert system in AFIT and available commercial packages.) Implementation was done in the C programming language, which although is not design specially for dealing with problems in the Artificial Intelligence (AI) field could be used with success. An integration with a graphics package and a central database was achieved.) The integrated system is currently loaded in an engineering workstation used in the Department of Computer Science and Electrical Engineering at the Air Force Institute of Technology.) Tests conducted with the system running in a personal computer Zenith 248 and compatible microcomputers under the Disk Operational System (DOS) version 3.2 proved the portability and efficiency of the expert system.

A user's manual is included for the operation of the InterConnect Expert system (ICE). Recommendations for future research are considered.

UNCLASSIFIED

AFIT/GCS/ENG/89J-1

AN EXPERT SYSTEM IN C FOR COMPUTER-AIDED
DIGITAL CIRCUIT DESIGN

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Systems

Jorge da Silva Santos, B.S.

Captain, BRAZILIAN AIR FORCE

June 1989

Accession No.	
NTIS CRA&D	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification:	
By:	
Distribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

Approved for public release; distribution unlimited



Preface

The purpose of this thesis was to design and implement a portable microcomputer rule-based expert system capable of detecting wiring errors in digital circuits constructed of TTL ICs. Computer listings of the system are not included in this document; but they can be obtained through the Air Force Institute of Technology, School of Engineering, Wright-Patterson AFB OH 45433.

This extensive, hard, but rewarding work could not be finished without a great deal of help from others.

I wish to express my sincere appreciation to my thesis advisor, Captain David W. Fautheree, who reviewed this work so many times for his patience and guidance. Thank you also to the members of my thesis committee, Captain Bruce L. George, who was always available and ready to show me the best way to improve my work, and Captain Wade H. Shaw, who gave me important and opportune suggestions to refine this document. A word of thanks is also owed to Captain Sue A. Ehrhart, for having helped me, in many ways, during the development phase.

My personal and special thanks to the Brazilian Air Force for putting me in touch with such advanced technology, that certainly will be used when I return to my country.

I want to dedicate this work to my family, my loving and often times neglected wife Gleide, without whose support and understanding this thesis would never have been completed; my daughters Giselle, Luciana, Renata, and Miriam for their sacrifice, patience, and love that made it possible for me undertake and complete this thesis.

Jorge da Silva Santos

Table of contents

	Page
Preface	ii
List of Figures	vii
Abstract	ix
I - Introduction	1-1
Background	1-1
Problem statement	1-6
Assumptions	1-6
Scope	1-7
Approach	1-7
Sequence of presentation	1-8
II - Literature Review	2-1
Expert Systems	2-1
Building Expert Systems	2-2
Applicable languages to expert systems	2-3
Analog and Digital Circuits	2-4
Current Technology	2-5
Summary	2-5

	Page
III - Requirements Analysis and Design	3-1
Introduction	3-1
Information Description	3-1
Information Flow	3-2
Information Content	3-4
Information Structure	3-4
Functional Description	3-6
Input phase	3-8
Processing phase	3-8
Output Phase	3-11
Performance requirements	3-11
Validation criteria	3-12
Performance bounds	3-12
Classes of tests	3-12
 IV - Detailed Design	4-1
Top level design	4-1
Module DEL FILES	4-1
Module OPEN TEMP-TXT	4-2
Module READ TEMP-TXT	4-3
Module SCAN FPKG	4-3
Module FILL ICETABLE	4-4
Module QUERY-PIN	4-4
Module MAKE PTICLIST	4-5

	Page
Module SCAN TPKG	4-5
Module FPKG or TPKG = TTL ?	4-6
Module ICERULE	4-7
Module PRINT-YMISTAKE	4-9
Module CIRCLEV	4-11
Module QUERY-FANOUT	4-12
Module CHIPLEV	4-13
Module GATELEV	4-15
Module QUERY-GATE	4-16
Module PRINT-RMISTAKE	4-17
Module WRAPUP	4-17
 V - Testing and Results	 5-1
Functionality Tests	5-1
Integration	5-1
Efficiency Tests	5-1
Portability	5-1
Independence of Commercial Package	5-2
Response Time	5-2
Space requirements	5-2
Results	5-3
Response Times	5-3

	Page
Space Requirements	5-14
Summary	5-15
VI - Conclusions and Recommendations	6-1
Conclusions	6-1
Recommendations	6-2
Appendix A: ICE Rules Set	A-1
Appendix B: User's Guide For ICE	B-1
Bibliography	BIB-1
Vita	VITA-1

List of Figures

Figure	Page
1 - 1 AFIT environment	1 - 5
3 - 1 Old ICE Interface	3 - 2
3 - 2 New ICE Interface	3 - 3
3 - 3 Input File Structure	3 - 4
3 - 4 Digital Circuit Example	3 - 5
3 - 5 Output File Structure	3 - 6
3 - 6 ICE System Conception	3 - 7
3 - 7 Production System Structure	3 - 10
4 - 1 Top Level Flowchart	4 - 2
4 - 2 SCAN FPKG Flowchart	4 - 3
4 - 3 SCAN TPKG Flowchart	4 - 6
4 - 4 Matrix of External Sources to TTL Connections	4 - 7
4 - 5 Matrix of TTL to TTL Connections	4 - 8
4 - 6 ICERULE Flowchart	4 - 9
4 - 7 CIRCLEV Flowchart	4 - 11
4 - 8 CHIPLEV Flowchart	4 - 13
4 - 9 GATELEV Flowchart	4 - 15
5 - 1 Performance New ICE vs Old ICE	5 - 4
5 - 2 Performance Characteristic of the New Ice (Time vs TTLs)	5 - 6
5 - 3 Correct Circuit with 2 TTLs / 2 Gates per TTL	5 - 7
5 - 4 Incorrect Circuit with 2 TTLs / 2 Gates per TTL	5 - 7

Figure	Page
5 - 5 Correct Circuit with 2 TTLs / 4 Gates per TTL	5 - 8
5 - 6 Incorrect Circuit with 2 TTLs / 4 Gates per TTL	5 - 8
5 - 7 Correct Circuit with 4 TTLs / 2 Gates per TTL	5 - 9
5 - 8 Incorrect Circuit with 4 TTLs / 2 Gates per TTL	5 - 9
5 - 9 Correct Circuit with 4 TTLs / 4 Gates per TTL	5 - 10
5 - 10 Incorrect Circuit with 4 TTLs / 4 Gates per TTL	5 - 10
5 - 11 Correct Circuit with 8 TTLs / 2 Gates per TTL	5 - 11
5 - 12 Incorrect Circuit with 8 TTLs / 2 Gates per TTL	5 - 11
5 - 13 Correct Circuit with 8 TTLs / 4 Gates per TTL	5 - 12
5 - 14 Incorrect Circuit with 8 TTLs / 4 Gates per TTL	5 - 12
5 - 15 Correct Circuit with 16 TTLs / 2 Gates per TTL	5 - 13
5 - 16 Incorrect Circuit with 16 TTLs / 2 Gates per TTL	5 - 13
5 - 17 Comparison of Disk Space Requirements(Old ICE vs New ICE)	5 - 14

Abstract

This thesis effort documents the design, development, implementation, and test of an expert system which decomposes digital circuits into subproblems in order to detect wiring errors, which consist of improperly connected gates, missing connections, and violation of fanout or race conditions. Information needed to connect chips together is viewed as knowledge base information for the expert system. Information such type as number of pins, value of each pin (input, output, power, ground, clock), fanout for a particular type of chip are retrieved from a central database where they are represented. The approach to this effort includes a examination of existing expert system in AFIT and available commercial packages. Implementation was done in the C programming language, which although is not design specially for dealing with problems in the Artificial Intelligence (AI) field could be used with success. An integration with a graphics package and a central database was achieved. The integrated system is currently loaded in an engineering workstation used in the Department of Computer Science and Electrical Engineering at the Air Force Institute of Technology. Tests conducted with the system running in a personal computer Zenith 248 and compatible microcomputers under the Disk Operational System (DOS) version 3 2 proved the portability and efficiency of the expert system. A user's manual is included for the operation of the InterConnect Expert system (ICE). Recommendations for future research are considered.

AN EXPERT SYSTEM IN C FOR COMPUTER-AIDED DIGITAL CIRCUIT DESIGN

I. Introduction

This chapter discusses the background of the problem addressed by this thesis effort. The problem statement is presented, followed by assumptions, scope, and approach taken for solving the problem. At the end of the chapter a synopsis of the remaining chapters is offered.

Background

Logic circuits for digital systems can be designed utilizing combinational or sequential logic. A combinational circuit consists of logic gates whose outputs are derived directly from the present state of the inputs without consideration of previous inputs. Unlike combinational circuits, sequential circuits, which employ memory elements in addition to the logic gates, present outputs are a function of the inputs and the state of the memory elements. The state of memory elements in turn is a function of previous inputs. Consequently, sequential circuits depend not only on present inputs but also on past inputs, and the circuit behavior must be specified by a time sequence of inputs and internal states.

Before it is ready for implementation, a circuit must be designed. The design phase includes decomposition into the input and output signals, boolean functions, and logic diagrams. There are several

methods for implementing digital circuits, one of these is the classical method, which tries to reach the following design objectives:

- minimum number of gates,
- minimum number of inputs to a gate,
- minimum propagation time of signal through the circuit, and
- minimum number of interconnections (Mano,1979:116-118).

Quite often is difficult to satisfy these constraints all the time.

According to the classical method, given two circuits performing the same function, one with fewer gates is preferable because it costs less, but this is not always true for integrated circuits.

Mano emphasizes that it may be more economical to use as many of the gates from an already used package, even if the number of gates is increased (Mano,1979). Moreover, some of the gate interconnections are internal to the chip and it is more economical to use as many of these interconnections as possible, in order to reduce the number of wires between pins. In integrated circuits, the number of gates does not determine the cost, so much as the number and type of integrated circuits used and the number of external connections utilized in the implementation. Considering all those mentioned facts, it is easy to conclude that designing digital circuits is not a trivial task; many factors come into play. Making the digital circuit design more complex is the fact that manually building the circuit on a circuit board is a tedious, time consuming task, and prone to errors.

Joseph Greenfield estimated that a circuit with five integrated circuits has "no better than 50% chance of working, due to wiring errors, when power is first applied"(Greenfield,1977). He states that

the probability of the circuit's working on the first try decreases as the complexity of the circuit increases.

Greenfield also states that the art of debugging, the process of locating and correcting errors, separates the expert engineer from the novice engineer, with the ability of the expert engineer being much greater than the novice engineer. To debug a circuit, it is necessary to know exactly what the circuit is supposed to do and the value for each pin that is wired. Comparing the results obtained from the circuit and the expected results should reveal the source of the errors as either wiring or logic. Wiring errors consist of missing connections, improperly connected gates, and violations of fanout, which is the number of standard loads that the output of gate can drive without impairing its normal operation. Logical errors are more complex to detect and require more detailed information about the chip performance. The way to find a solution to the problem is to functionally decompose it (Descartes' principle of analysis); then by assembling the partial solutions to a problem, the problem is solved incrementally (Descartes' principle of synthesis).

Expert systems are especially successful with problems that can be decomposed into subproblems. An expert system, in general, is composed of three main parts: knowledge base (expertise), inference engine (deductive part), and user interface. The knowledge base provides the material for the inference engine. As questions are posed by the user, the inference engine infers partial solutions from the knowledge base. The sum of the partial solutions provides the completed answer.

A rule-based expert system is one in which the knowledge base is represented by rules, with each rule having a premise (also called LHS-left hand side) and a conclusion (also called RHS-right hand side).

Another critical part of an expert system is the user interface. The interface should provide a friendly development environment to the user, offer easy access to stored data, and give rapid of problem resolution.

Many of the current uses of expert systems are in computer aided design (CAD) work at the component level. Many assume that the circuit is wired properly and limit the aid to component layout or selection. Using an expert system to debug a digital circuit at the gate level seems to be a quite novel idea.

The Air Force Institute of Technology (AFIT) School of Engineering had developed a prototype computer-aided design (CAD) workstation that is built of a simulator (LOGSIM) and a VAX based expert system (ICE). LOGSIM is a pin-level logic simulator which verifies circuit performance. Although ICE (InterConnect Expert) is used to identify wiring errors, ICE does not detect logic errors. Through the use of these two independent systems, a designed circuit can be tested without having to wire the actual components for verification (Estes, 1986).

The tools which are available are hosted on separate machines. To use the tools, the designer has to learn how to operate each system. In addition to the fact that the VAX is isolated from the lab, the expert system does not have a library or common interface, lacks responsiveness and also limits the number of integrated circuits that can be used (Wagner, 1987).

All students attending the AFIT School of Engineering must have a background in digital electronics. Supporting this requirement, an initial course in discrete component design and analysis using transistor-to-transistor(TTL) devices is presented. Although, the system in AFIT should be a basic tool for students attending that course, its location in more than one environment (VAX and workstation) greatly reduced the system's utility.

In order to make the system useful for students attending AFIT, it was necessary put the entire system in one environment. In 1987 a integrated system was developed on a microcomputer. The system was composed of the simulator (LOGSIM) developed by Captain Wayne DeLoria; the graphics part, developed by Captain Charles A. Adams; and the expert system, developed by 1Lt Steven Wagner. Figure 1-1 illustrates how the graphics part, LOGSIM and ICE are combined into one environment.

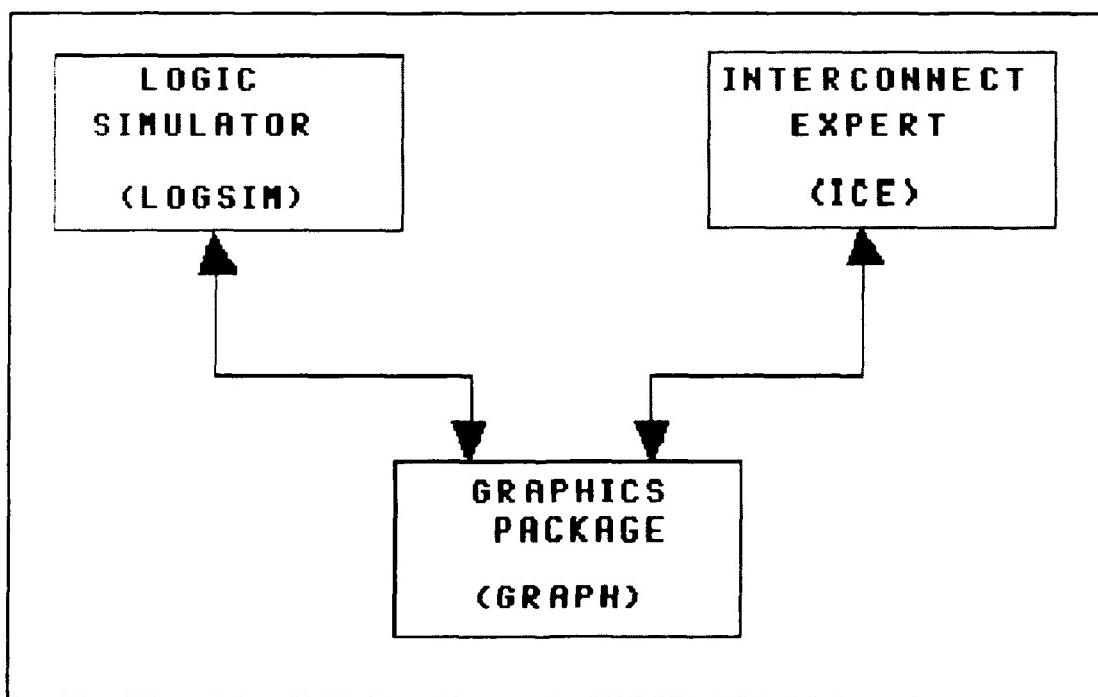


Figure 1-1 AFIT environment

The expert system was built using the development environment in the commercial package GURU. The system on the microcomputer proved to be very useful, but still some enhancements were necessary.

Problem statement

This thesis addresses some problems in the ICE installed on the microcomputer. The first problem is to make the expert system portable, i. e., capable of installation in a different machine from the one in which it has been developed without modifications on the code of the system. The second problem is to reduce the size of the expert system due to constraints on the microcomputer memory. The third problem is to integrate the expert with the microcomputer version of the LOGSIM, the graphics package, and a database. The effort of this thesis is to design, develop, test, and evaluate a portable microcomputer rule-based expert system capable of detecting wiring errors in digital circuits constructed of TTLs ICs using a commercial high order language.

Assumptions

Rule-based expert systems can be built with forward chaining or backward chaining. In a forward chaining system; when the premise clauses match the situation, the conclusion clauses are asserted. In backward chaining system, the goal clause is matched with the conclusion of rules. The matched rules premises then become new goals. For solving the problem, forward chaining was assumed as best choice, because the branching factor, i.e. the average number of nodes that can be reached directly from a single node, is lower than using backward chaining.

The programming language "C" is highly portable, powerful, and can be used to code an expert system with relative ease. "C" was determined the ideal language to codify the expert system.

Scope

In the construction of the expert system, special attention will be given to the database interface, to the user interface, and to the graphics package interface. The entire system should be able to be implemented on a microcomputer. An evaluation of the system, in relation to the previous one, will be conducted by means of submitting several circuits to both systems and recording their performances as well as comparing size of code, and size memory used to perform their tasks. That evaluation will assess the benefits that the new system will bring.

Approach

This effort was accomplished in five stages. The first stage was the analysis of the existing system that pointed to the weak points of that system such as: poor response time, a complete dependency of a copyrighted commercial package that implied a very limited utilization of the system because few sites were authorized to use that package, a large disk space necessary to accommodate the code, the large amount of memory necessary to accommodate the executable code. The second stage was to take the result of the analysis conducted in the previous stage and the current literature about CAD development related to expert systems in order to obtain the directives that led to the construction

of the expert system prototype. Also, in this stage, meetings were held with Capt Ehrhart and Capt Matuszek, who were making concurrent enhancements to the environment. In those meetings it was established how the common information would be manipulated and displayed. Also, a consensus about the programming language was reached. In the third stage the prototype expert was designed, developed in the C programming language, and tested stand alone. The fourth stage included an integration with the database developed by Capt Ehrhart and with the graphics package, testing the expert system performance after integration, and final evaluation of the expert system. The delivery of a final expert system, derived from the refinement of the prototype, as well as the documentation necessary to maintain and operate the system, took place in the fifth stage.

Sequence of presentation

Chapter 2 covers the research that has been done into the expert systems field to support the effort of this thesis, and also shows how the result of the literature review has been used to produce the new prototype of ICE. Chapter 3 identifies the necessary data elements, how they were organized, giving the criteria utilized for such construction of ICE. Chapter 4 describes each module of ICE, and the method of operation of each module which resulted from the implementation phase. Chapter 5 discusses the outcome of tests conducted during all the implementation phase of the new prototype of ICE, as well as the analysis of those results confronted against the results obtained when running the old version of ICE, the one that relies on GURU. Chapter 6

contains final directives for future research and improvement in the field covered by this thesis. Appendix A encapsulates the set of rules used by ICE. The subject addressed by Appendix B is the user's guide for the ICE.

II - Literature Review

This chapter briefly traces the expert system history. The importance of the book Building Expert Systems in the expert system era is emphasized. A discussion about applicable languages is also done. Next there is an introduction to analog and digital circuits, a description of the effort of AFIT students who applied expert systems to analyze digital circuits, the current technology, and a summary.

Expert Systems

Destined to drastically reduce the price of computers, early research in electronic device fabrication brought about the microchip technology. Meanwhile the software specialists, people responsible for designing and building programs to control computers, were working in the software area. They were not trying to discover a new way to encode information in microchips; they were laying the basis of a field of computer science known as artificial intelligence (AI).

The main idea of the AI scientists was to provide some sort of thinking to the computer programs, so the computer could solve problems in the same way that a intelligent human being would do it. In the 1960's, AI scientists tried to simulate a complex method of thinking, that phase yielded some progress but no breakthroughs occurred.

Later, AI scientists tried another manner to provide intelligence to a computer. They concentrate on techniques like representation, a way to formulate a problem in order to facilitate its solution, and search mechanisms to reduce the time and utilization of computer memory when

solving a problem. Again some success was obtained but no breakthroughs occurred.

In the early seventies the AI scientists realized an important point. The power for solving a problem that a computer should have was not provided by the formalism and the inference schemes that the computer employs; rather, the power is a consequence of the knowledge that the program possesses. This concept was embraced and stated as:

"To make a program intelligent, provide it with lots of high-quality, specific knowledge about some problem area" (Waterman, 86).

This breakthrough made possible the development of special-purpose computer programs, systems that were expert in some narrow problem area. These programs were called expert systems, and a new field began (Waterman, 86).

Building Expert Systems

The efforts of more than forty AI scientists, that are compiled in the book Building Expert Systems, which organizes the technical state of art and describes several possible expert-system-building techniques when solving a common problem.

The book discusses the interaction between the knowledge engineer, expert system builder, and one or more human experts in some specific problem area. The knowledge engineer has as his main task extraction from the human experts their methods and strategies for problem solving, and construction of the expert system based on this knowledge. The result should be a computer program that solves the problem in a manner close to that one used by the human experts.

Most AI scientists, agree that the main parts of an expert system, are the knowledge base and inference engine.

Mike Van Horn (Horn,86), explains both, the knowledge base and inference engine, and concludes that the knowledge engineer derives rules from the experts' statements and separates the rules into knowledge rules and inference rules. Some AI scientists prefer "reasoning" instead of "inference", which is considered a fancy term. Horn admitted also that the knowledge engineer, based on his experience in dealing with expert systems, adds some inference rules. The knowledge rules become the knowledge base and the inference rules the inference engine.

Applicable languages to expert systems

Among programming languages, the most appropriate to deal with an expert system is LISP, a symbol manipulation language. That is, a language designed for representing and manipulating complex concepts. But LISP with its great power and flexibility brings a high cost. First, in order to achieve the enormous power and flexibility, LISP requires a complex and convoluted code, which is difficult to program. Also LISP programmers are rare and expensive. Second, more extensive code requires more computer processing time, and more processing time means that more money is spent when the system is running. Third, the more extensive is the code, the more extensive is the amount of memory required for running the code.

Because the language most appropriate to handle expert systems presents so many drawbacks, the AI scientists turned their attention to

other languages that were not addressed to expert systems, and among those languages, "C" presents excellent performance. Programming in "C" is generally easy, it requires few lines of code, and programs coded in "C" have great portability. That is, when moved to a computer different from that one where they have been developed they require few or no adaptations. The drawback of "C" is the necessity of having all the basic functions, such as `open_file` and `display_character`, developed by the programmer; but even this drawback is not serious because there are many C-programmers, they are not expensive, and many basic functions are already commercially available, usually known as "C libraries".

Analog and Digital Circuits

There are two types of circuit, digital and analog. The most frequently used is digital. Several researches have reported various techniques for working with analog or digital circuits. Lieutenant Steven Wagner in his thesis (Wagner,87) reports on the efforts of AFIT students who examined the effects of an expert system on digital circuits. Those students designed and built a system called InterConnect Expert (ICE), which was an expert system used to identify wiring errors in a digital circuit by using a rule-based system. The circuit was checked without having to wire the actual components of circuit.

Later, Lieutenant Wagner introduced significant modifications in that project in order to improve the performance of the InterConnect Expert. This experience is reported in Wagner's thesis (Wagner,87).

Current Technology

The current technology available in the market is limited. Wagner pointed out two of the more advanced systems, and from that time to now, they are still the most advanced systems. They are: the HIWIRE, addressed to IBM PC, with the capacity to utilize up to 700 TTL components when designing a digital circuit, and the HP Electronic Design Circuits, developed by Hewlett Packard (HP), addressed to the HP workstations.

Summary

Expert systems are excellent tools for checking digital circuits without having to wire the actual components.

Although the most specific language to code an expert system is the LISP, its utilization is not encouraged due to the fact that is difficult to code in LISP, also because it is not easy to find a LISP programmer, and furthermore because LISP requires large amount of memory. On other hand, is very easy to find a "C" programmer. "C" is a multipurpose language, very powerful, easily learned and highly portable which makes it the language suitable to be used in codification of an expert system.

The commercial packages available in the market do not fill the needs of AFIT, due in part to the necessity of specific equipment to operate them, and also in part to the necessary licenses for site operation. Which in both cases imply cost and reduced flexibility in the operation of those packages.

The expert system developed in AFIT needs some improvements such an increase in the processing speed, and especially a divorce from the commercial package GURU on which the expert system was developed on.

The natural answer to the AFIT needs, in face of what resulted from the literature review, was to recode in "C" a new expert system that was completely independent from any commercial package, and which could bring an improvement in the processing speed.

III - Requirements Analysis and Design

Requirements analysis is the first technical step in the software engineering process. It is at that point that a general statement of software scope is refined into a concrete specification that becomes the foundation for the development phase (Pressman,87).

This chapter reports the results of the requirements analysis conducted in the ICE development effort.

Introduction

The effort of this thesis, as described in the Problem Statement section, is to design, develop, test and evaluate a portable microcomputer rule-based expert system capable of detecting wiring errors in digital circuits constructed of TTL ICs using a commercial high order language. The expert system should present a better response time than the older system. Also, the utilization of main, and secondary memory should be reduced.

One of the constraints for developing the expert system is the microcomputer in which the expert system should operate. It is a microcomputer compatible with Zenith Z-248 with 640 Kbytes of main memory, graphics card, and a hard-disk drive.

Information Description

This section provides a description of the information flow, information content, and information structure.

Information Flow

The old expert system relied heavily on the interface to the graphics package. This interface is shown in Figure 3-1. The graphics package placed an ASCII file in the Disk Operating System (DOS) environment, then invoked GURU. Guru automatically executed the expert system by invoking a startup file which acted as a mainline program. The expert system used the input file to start the processing and generated an output file, containing any circuit errors, upon completion. The output file signaled the end of execution and control was returned to DOS, which invoked the graphics package (Wagner,87).

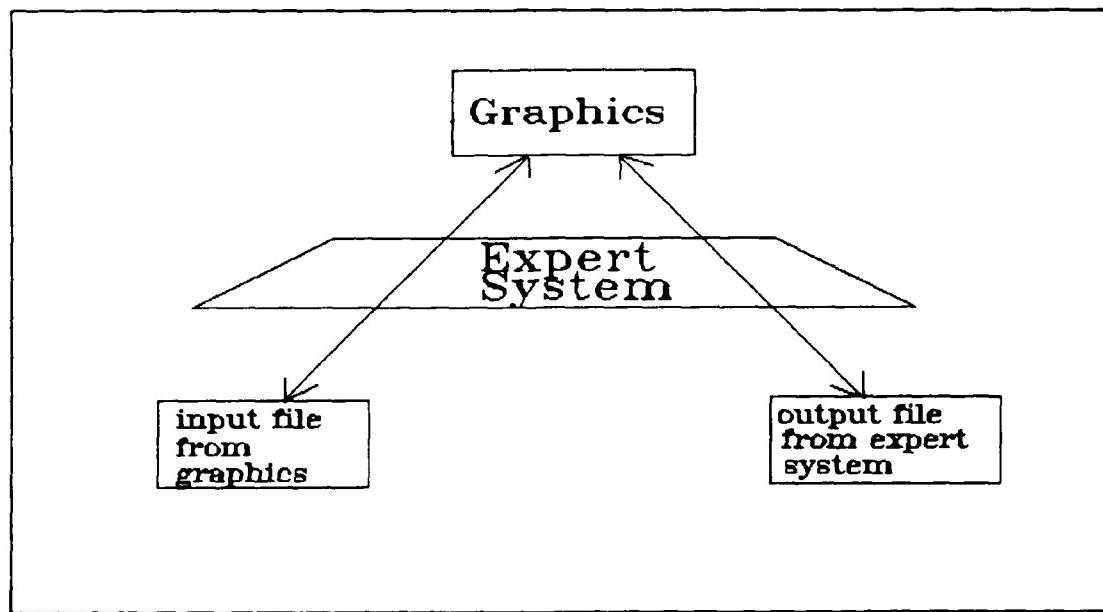


Figure 3-1 Old ICE interface

The old ICE system contained hardcoded, information pertaining to TTLs. This information was condensed in a table called Engineering Workstation Chip Library. That table could only be "seen" and used by the old ICE system. But ICE was, and still is, part of a larger system. So, any modification forced updates in more than one place throughout the larger system, because the other integrated parts also had their private tables of TTL information. With the improvement that resulted from using a central database, used by all integrated parts of the larger system, the ICE interface assumed a new form as shown in Figure 3-2. The basic idea of the new interfaces is the same as the one posted in the old system. But now, the information about TTLs are extracted from a central database, and ICE no longer uses GURU.

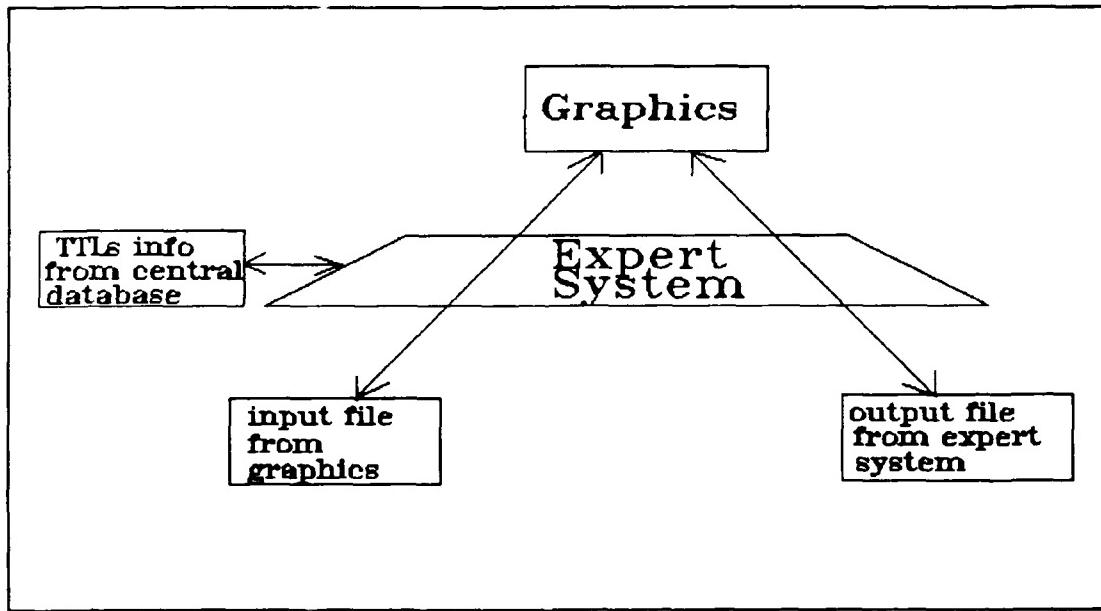


Figure 3-2 New ICE interface

Information Content

The input file from the Graphics interface, a flat file called TEMP.TXT, contains all connections used to build the digital circuit. Connections can be of two types: TTL to TTL, or TTL to external source.

The output file from ICE, a flat file called ICEOUT.TXT, contains all flagged missing connections, at least one pin of an used gate had been not wired, as well as all the flagged wrong connections, those that are not allowed in digital circuits - e. g. input of a gate to the input of another gate. The output file can have only missing connections, or only wrong connections, or missing and wrong connections, or neither missing nor wrong connections, when then the sentence "No output from ICE" is posted in the output file.

Information Structure

The input file structure, represented in Figure 3-3, is made in such way that each line of file stands for a connection in the circuit. The connection record is divided into two parts: FROM and TO. FROM registers from where the connection is originated, and TO brings all information of where the connection ends. Each part has the following information: unique identification for the package (TTL or external source), package type, pin of the package where the connection starts or ends.

FPKG	FTTL	FPIN	TPKG	TTTL	TPIN
------	------	------	------	------	------

Figure 3-3 Input file structure

For example the following shows the input file corresponding to the digital circuit shown in the Figure 3-4.

```
T001 07419C 08 G000 GROUND 00  
P000 OPOWER 00 T001 074193 16  
T001 074193 04 C000 OCLOCK 00  
T002 007400 14 P000 OPOWER 00  
G000 GROUND 00 T002 007400 07  
T001 074193 13 T002 007400 13  
I001 0000i1 00 T002 007400 12
```

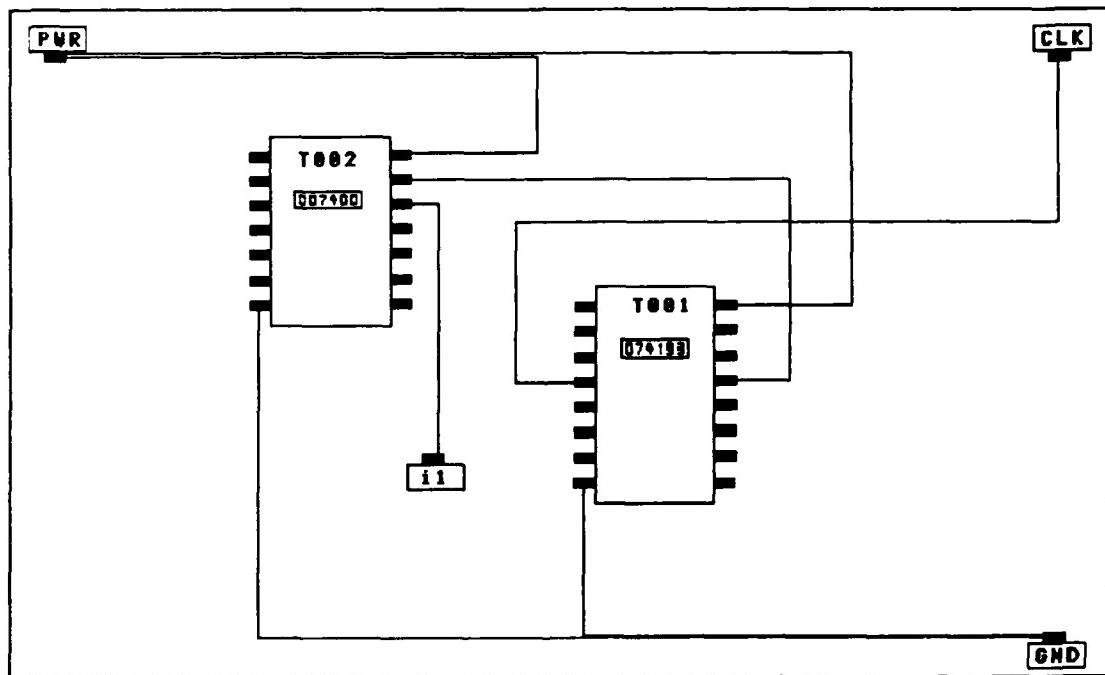


Figure 3-4 Digital circuit example

The output file structure, as can be seen in Figure 3-5, is simple. Each line brings a description of the problem found, and the location of the problem. Each region, missing links and questionable links, is preceded by pertinent heading.

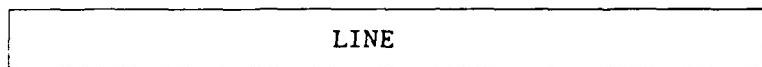


Figure 3-5 Output file structure

An example of an output file is below depicted.

Example:

The link(s) between the following pin locations are questionable:
two INPUT's are tied together -- pin 01 of T002 and pin 09 of T001
package T001 has a race condition present between pins 01 and 02

There are also missing connections at the following pin locations:

package T001 a 7400 is missing an OUTPUT at pin 03
package T001 a 7400 is missing an OUTPUT at pin 08
package T002 a 7400 is missing an INPUT at pin 10

Functional Description

The new InterConnect Expert system, like its old version, presents three distinct phases, with each phase performing the necessary transformations to achieve the expected result. Figure 3-6 shows how, conceptually, the system is divided.

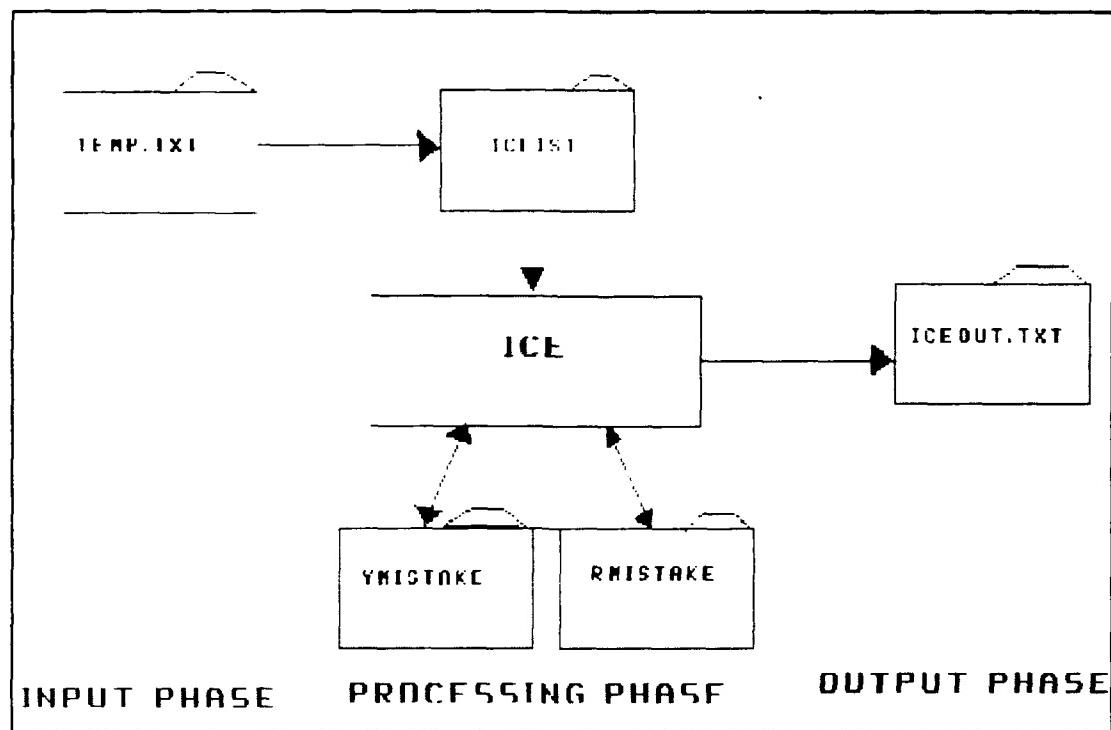


Figure 3-6 ICE system conception

Although the division of the ICE system in three phases follows the same pattern applied in old version of the system, the way that each phase is performed differs substantially from the previous model.

For instance, in the new version a field was suppressed in TEMP.TXT. That suppression brought a considerable economy in the processing time because less time is spent in reading and writing. Furthermore the field was used for comments, which often represented forty percent of the record length.

The manner in which the processing phase is conducted - without GURU, "lighter", and much less avid for memory and time - contributed to an

upgrade in the response time. A comparison between the performances of the new and the old version of ICE is presented in Chapter 5.

In the output phase, a better strategy related to the intermediate files Ymistake and Rmistake resulted in a better use of the memory and an abbreviation in the composition time of the output file. Instead copy them all into a third file a combined action of renaming and appending was used.

Input phase

The system makes sure the file TEMP.TXT, which contains all links designed in the digital circuit, is present. If the file is present, it is opened and made available for the processing phase. If the file is not present, the ICE is halted after issuing a message which alerts the absence of the input file.

Processing phase

The processing phase starts by building from the TEMP.TXT a larger table, ICETABLE, which, besides the previous information, contains the gate related to each pin in the package as well as the value associated to the gate (input, output, clock, ground, power, non connecting). This additional information comes from the central database, which is accessed by means of a specific query. After that, the processing phase continues by constructing a file, ICLIST, containing the links in it that involve at least one TTL. Continuing the processing phase a set of rules is then applied to ICETABLE to determine questionable links. A rule is a conditional statement of two

parts: the first part, comprised of one or more IF clauses, establishes conditions that must be satisfied and the second part, comprised of one or more THEN clauses, is to be acted upon.

In case of the existence of one or more questionable link, the file Ymistake is constructed. Once the ICLIST is completed, it is sorted by a composed key made of TTL type, package identification, gate and pin number. Once finished, the file is analyzed at circuit level, chip level, and gate level for determining if there are missing links or fanout errors. In these tasks, several accesses to the central database are made in order to validate the connections found in the circuit.

In case validation failure, implying existence of missing links or fanout errors a file, Rmistake, is built to conclude the processing phase.

The set of rules early mentioned is part of the production system utilized to detect questionable links. A product system consists of a collection of production rules (rules in the form of an IF-THEN or CONDITION-ACTION statements), together with a database of "state information" and a procedure for invoking the production rules. The overall structure of a production system is shown in Figure 3-7.

The problem addressed by this thesis was solved by making use of forward chaining (inference engine where the IF portion of rules are matched against facts to establish new facts) and a production system where the base of production rules was named ICERULE (described in more detail in Chapter 4), the database of state information representing the circuit design was taken from ICLIST, and with the control scheme put into the main procedure. The control scheme works scanning through

a list of production rules. The condition part of each rule is tested in turn, until one of the condition is found to be true or there is no more condition to test. When one is true, it is said that the production rule "fires" or "triggers". Then, the action for that rule is executed.

With ICE implemented in this manner, and because the production rules hold for all TTL, it is possible to analyze any TTL provided TTL data is in the central database. This means that is possible to add TTLs to the system, and analyze them without any required modification in ICE.

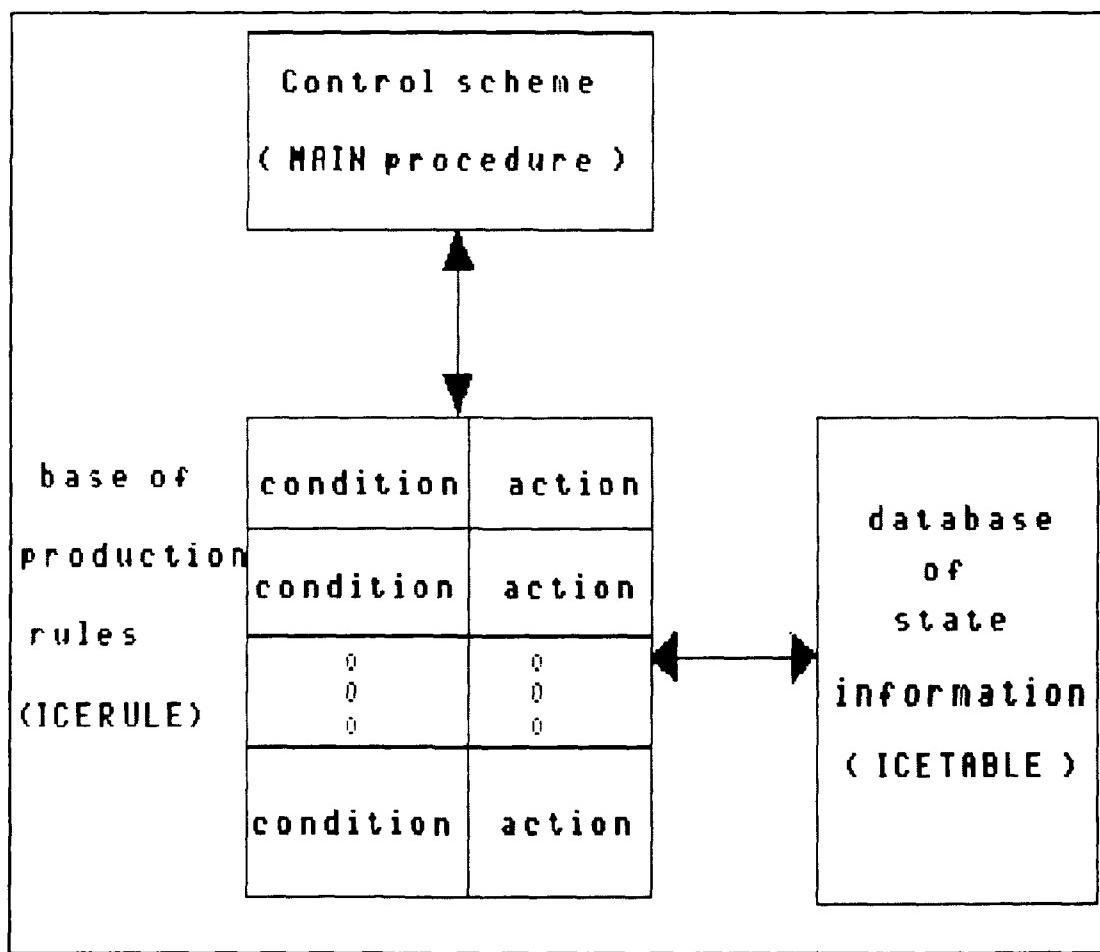


Figure 3-7 Production System Structure

Output Phase

The output phase consists in the construction of the output file. The construction of the output file can be carried out in one of the following four possible manners:

- 1 - if Ymistake and Rmistake exist, Ymistake is renamed ICEOUT.TXT and Rmistake is appended to it,
- 2 - if only Ymistake exists, it is renamed ICEOUT.TXT,
- 3 - if only Rmistake exists, it is renamed ICEOUT.TXT,
- 4 - and if neither Ymistake nor Rmistake exist, ICEOUT.TXT is created having just one record that is the sentence "No output from ICE".

Performance requirements

The performance requirements addressed in this thesis effort are those mainly concerned with the system time response that is considered extremely long in the old system. The goal was to reduce the response time to at least half that on the old system. A desired quality of ICE was to be completely independent of any commercial package, which would help the ICE distribution into the educational system. Special attention to the utilization of memory was taken, and since ICE is not a general system, but a very specific system, it could be reduced either in disk space as well as in main memory space.

Validation criteria

The special purpose of validation criteria is to make possible recognize if the implementation was well succeeded or not. In this task aspects as performance bounds, classes of test, expected software response are examined.

Performance bounds

The performance bounds imposed to the new iCE system were mainly those related to the hardware used, a microcomputer Z-248 with 640 Kbytes of main memory, graphics card, and a 20 Mbytes hard disk.

The time response of the new system could in the worst case be equal to the time of the old system, but could never exceed that time.

Classes of tests

The tests were divided in classes in such way that all the rules were fired at least once.

The same circuit was submitted to both systems, the old and new, and the response times were registered, compared and mapped into a graph for analysis.

The results of ICE tests, the expected ICE response, as well as the comparison graph between the old and new system are shown in the Chapter 5, that follows.

IV - Detailed Design

The purpose of the sections of this chapter is to show details of the system design. To support the explanations, flowcharts are incorporated into the chapter. In a flowchart, a box with double lines at its left and right ends, means that the particular box, most of the cases representing a procedure, will be decomposed later in the chapter. Boxes with single lines are not further decomposed. Diamond shapes represent points where a decision has to be taken, with Y meaning the answer "yes" to the question enclosed in the diamond shape, and N standing for "no". The text inside diamond shapes reveals the kind of test used, and inside a box names represent the procedure or set of statements utilized in that part of the design. Arrows show the direction in which the flow of information is carried out.

Top level design

The flowchart that best describes the system is shown in the Figure 4-1. Those modules depicted in the figure are described in more details in the sections that follow.

Module DEL FILES

The module DEL FILES represents a deletion of the files iceout.txt, ymistake, and rmistake that could have been resulted from previous ICE execution. In this way it is ensured that no previous results interfere with the results obtained in the current ICE execution. This module was implemented as a set of statements in the main procedure.

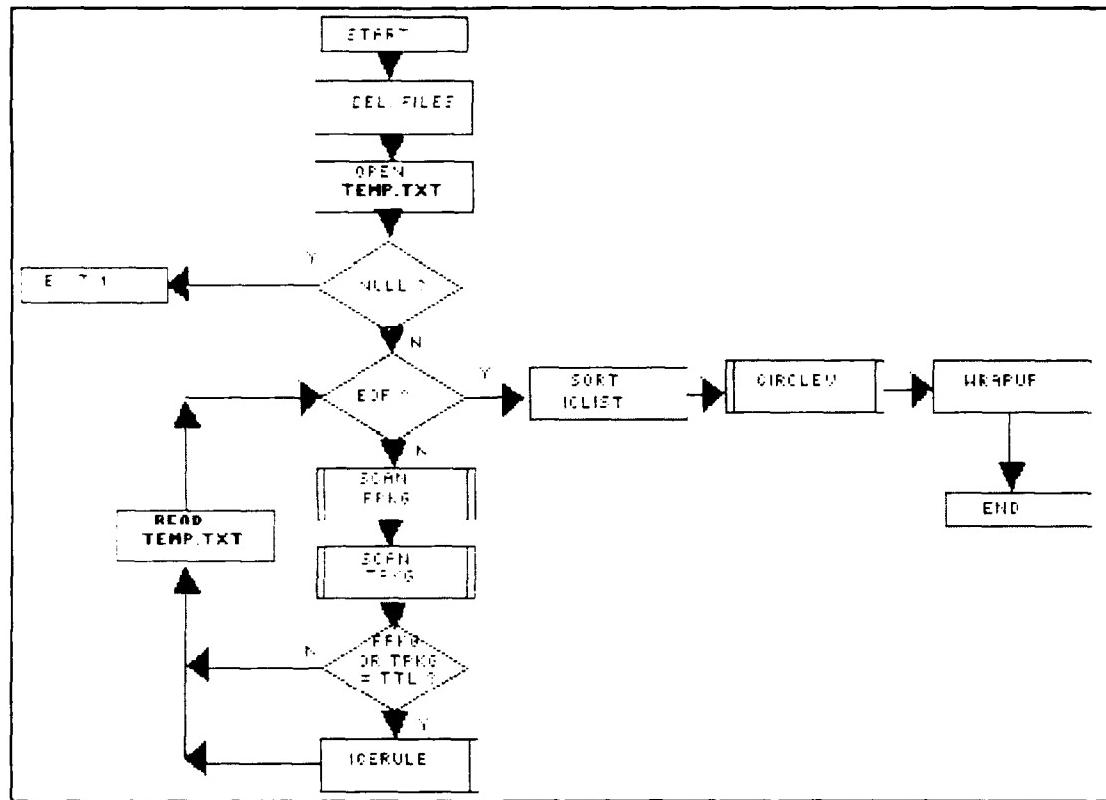


Figure 4-1 Top Level Flowchart

Module OPEN TEMP-TXT

In this module, an attempt to open the input file TEMP.TXT is made. Any case of failure in the file opening, meaning that the file does not exist or has any sort of reading problems, is always flagged by module "NULL ?". If there is a failure the module "EXIT 1" issues a message to the user warning him about this fact, and the system ends its processing because it is totally impossible to start the analysis of the circuit without the input file. The modules "NULL ?" and "EXIT 1" are shown in the Figure 4-1.

Module READ TEMP-TXT

In case of a successful opening, the system keeps reading the file until end-of-file (EOF) is reached. Between two consecutive readings, and between the last reading and EOF the following tasks are done.

Module SCAN FPKG

This module scans the FROM part of input file structure, to obtain the kind of component used as "from package" (FPKG). A schematic flow is shown in the Figure 4-2.

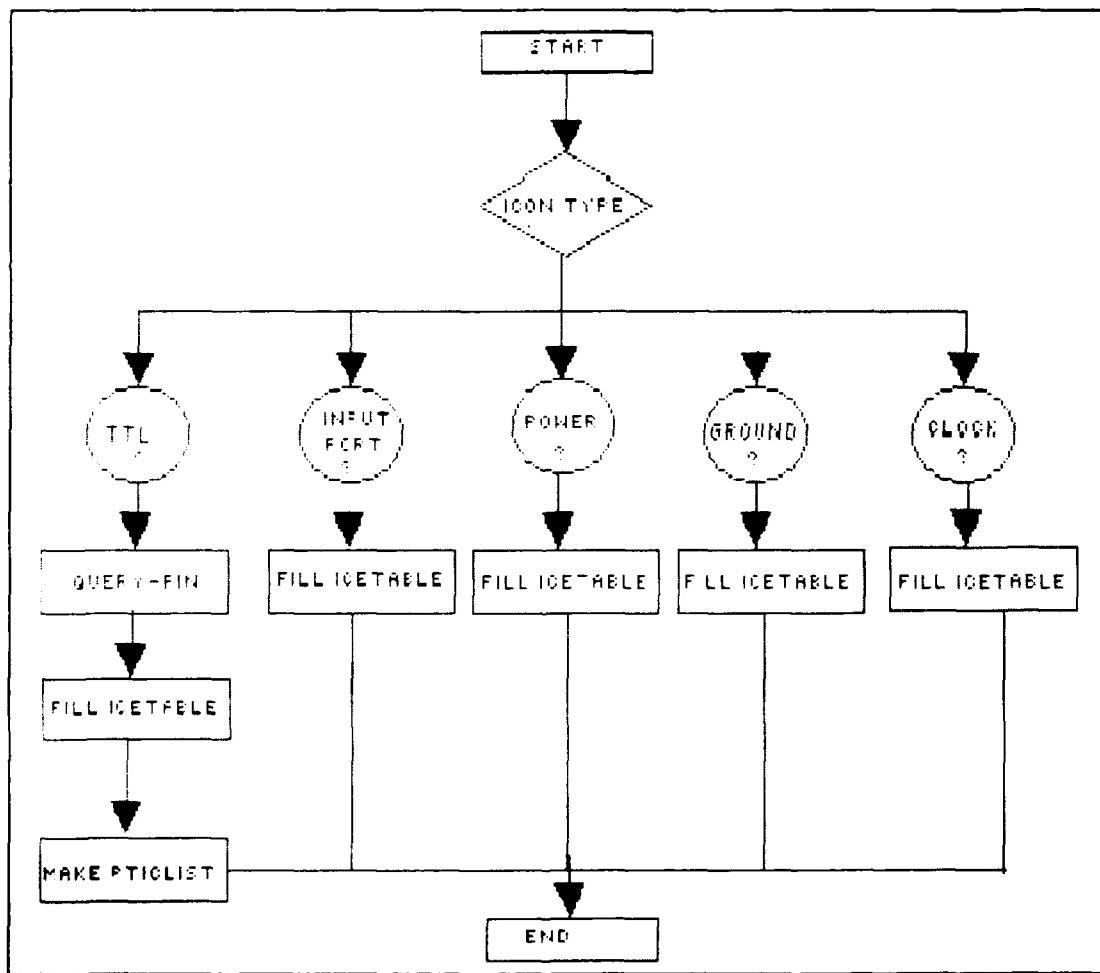


Figure 4-2 SCAN FPKG flowchart

Depending on the icon type used as "from package" different action is taken by the module FILL ICETABLE.

Module FILL ICETABLE

The set of statements represented by FILL ICETABLE has as its main purpose to complete the information about the component utilized putting a new piece of information into the structure ICETABLE. If the component is not a TTL such new piece of information is composed of the word ICON, and depending on the kind of icon INPUT, POWER, GROUND, or CLOCK. If the component is a TTL, the structure ICETABLE is completed with the name of the gate (e. g. A, B, C, etc) and with its respective value (e. g. INPUT, POWER, CLOCK, OUTPUT, GROUND). The additional information about the TTL comes from the central database accessed by the module QUERY-PIN.

Module QUERY-PIN

This module interacts with the central database searching for the attributes of a specific pin of a specific TTL. The searching is carried out on the relation PINS of the database. As an access key, a combination of TTL type (e.g 7400) and pin number (e.g 14) is passed to the database. The module returns a pointer to a memory location where the information, name and value, about the gate associated to this pin can be found.

Module MAKE PTICLIST

Making this module a procedure would force the utilization of a large number of parameters, or utilization of global variables. Since none of those options presented a high cost benefit rate ,this module was implemented as a set of statements in the main procedure. This approach achieved the same result that would be reached with the other options but with a much less complex code . The main module function is to construct, in memory, pointers to the structures which will serve as basis for the file ICLIST. Those structures contain the information about the FROM part of the input file structure (what means TTL type, package identification, and pin number), and the information retrieved by QUERY-PIN from the central database (gate name and gate value related to the pin number).

Module SCAN TPKG

This module as shown in the Figure 4-3, is a repetition of the module SCAN FPKG, but now the scanning is made on the "TO" part of the input file structure. Its main purpose is to obtain the kind of component used as "to package" (TPKG). The explanations given for FILL ICETABLE, QUERY-PIN, and MAKE PTICLIST in the module SCAN FPKG are also applicable to their counterpart in the module SCAN TPKG. Considering the similar behavior encountered in SCAN FPKG, and in SCAN TPKG no further discussions about the second will be conducted.

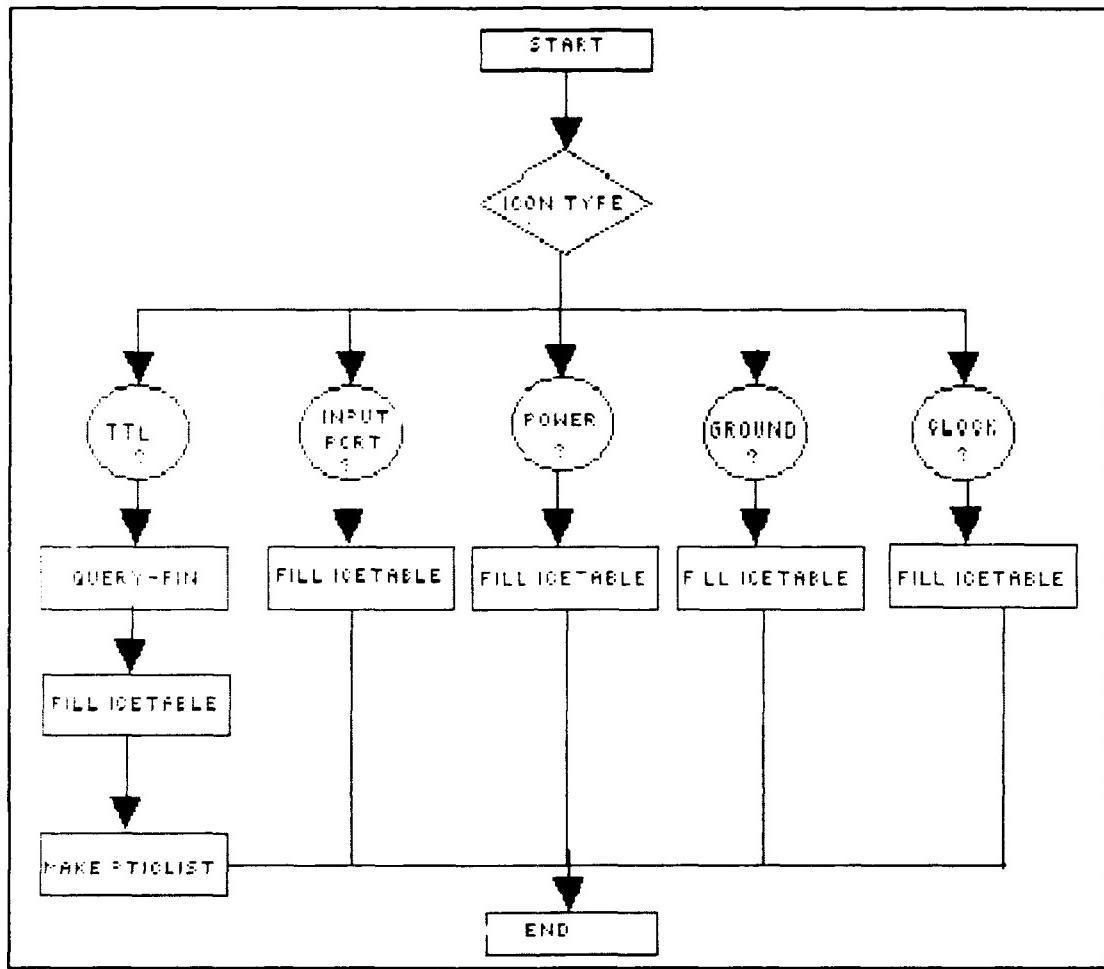


Figure 4-3 SCAN TPKG flowchart

Module FPKG or TPKG - TTL ?

At this point a decision is made to know if at least one of the components used in the link is a TTL. If that is not true a new record from the input is read. If at least a TTL is present on the link the procedure ICERULE is invoked, and after it is finished a new reading is provided .

Module ICERULE

This module applies the set of rules listed in the appendix A to each link that contains at least one TTL in order to find out any possible questionable link. The set of rules used is based in the same principles used to define the set of rules in the older version of ICE (WAGNER,87). Since two kinds of connections are use in a circuit (external sources to TTL, and TTL to TTL), two matrices of information were built.

The external sources that can be connected to a TTL are: INPUT port, POWER, GROUND, and CLOCK. Those external sources can be connected to the following kinds of TTL pins: INPUT, POWER, GROUND, CLOCK, and OUTPUT. In the matrix, shown in Figure 4-4, each connection is labeled as legal or illegal (L or I). Its better to point out that some connection marked as illegal could be used in a circuit and the circuit would function. For instance, although a connection between a external source CLOCK to a TTL pin INPUT could be legitimate, it is seen as illegal by ICE.

		TTL values				
		Input	Power	Ground	Clock	Output
External Connections	Input	L	L	L	L	I
	Power	L	L	I	I	I
	Ground	L	I	L	I	I
	Clock	I	I	I	L	I

Figure 4-4 Matrix of External Sources to TTL Connections

The second kind of connection, TTL to TTL, was treated in analogous manner as the external connections to TTL. Figure 4-5 shows the defined matrix for connection TTL to TTL.

		TTL values				
		Input	Power	Ground	Clock	Output
TTL values	Input	I	I	I	I	L
	Power	I	I	I	I	I
	Ground	I	I	I	I	I
	Clock	I	I	I	I	I
	Output	L	I	I	I	I

Figure 4-5 Matrix of TTL to TTL connections

The module ICERULE compares the current Icetable structure against the rules, and if one of the rules is fired a call to module PRINT-YMISTAKE is done to put the message corresponding to the event in the YMISTAKE file. A flowchart of the module is depicted in the Figure 4-6.

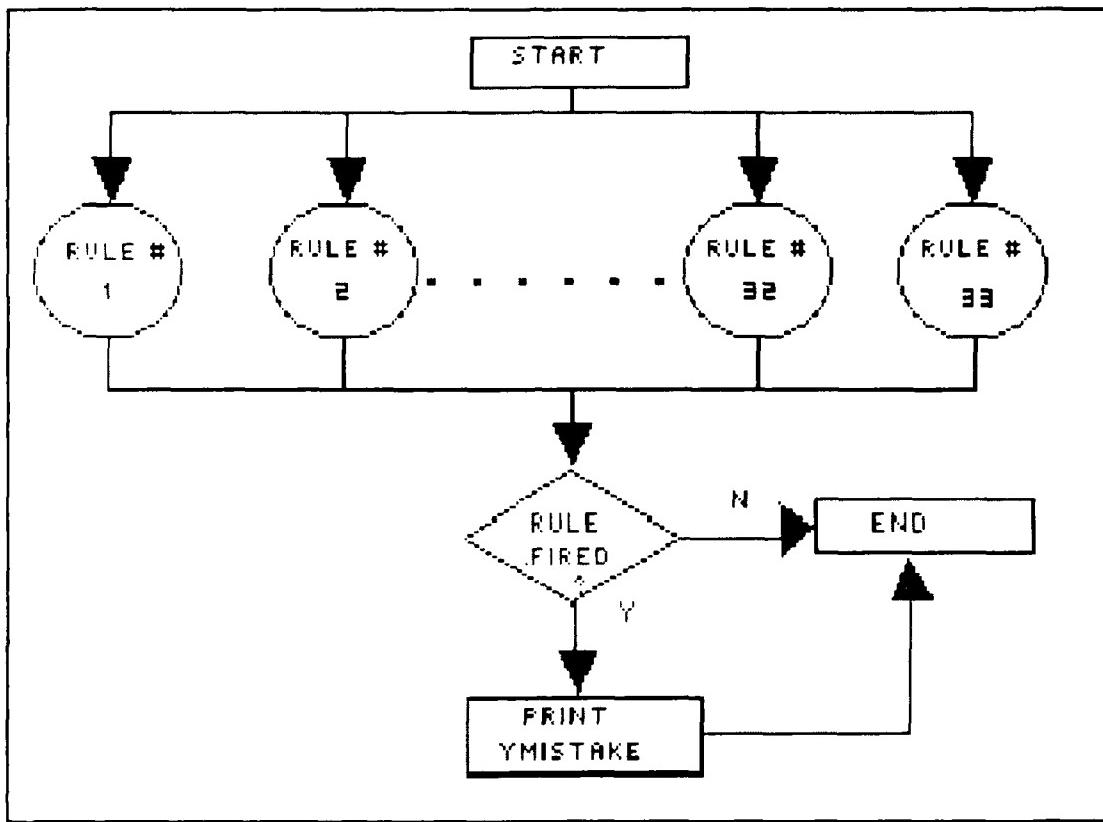


Figure 4-6 ICERULE Flowchart

Module PRINT-YMISTAKE

This module is responsible to put the questionable link message into the file Ymistake. To perform its tasks, this module takes from ICERULE the current ICETABLE structure and the rule number which flagged the questionable link. The message corresponding to the rule number is then recorded onto Ymistake.

After EOF of the input file is reached, we have the following situation in the system:

- a) all questionable links in the circuit were detected and the correspondent messages are in the file Ymistake.
- b) the memory contains a set of ICLIST pointers which will be used to construct the ICLIST file, such file is used to decompose the circuit into chip and gate levels.

Constructing the file ICLIST requires writing on disk the memory contents pointed to by each ICLIST pointer. But the order of the pointers on the memory is dictated by the order that the user put the links in the circuit. Trying to improve the efficiency of the system when decomposing the circuit at chip, and gate levels a particular order was adopted, and that adoption implies sorting the content of the file ICLIST.

The sorting could be done in one of two ways:

- a) write the pointer's contents on the disk file, and then sort the file;
- b) sort the pointers on the memory and then with the pointer's contents already sorted make all the writing into the disk file.

The option of sorting the pointers before writing was adopted because it proved be the faster, and more efficient than sorting the file on disk which would imply many accesses to disk, and thus time consuming. In order to perform the sorting the procedure SORT_ICLIST was built.

Module SORT-ICLIST

This module uses the merge-sort algorithm to order the pointers according to TTL type, package identification, gate identification and pin number.

Module CIRCLEV

This module decomposes the entire circuit into chips, for further analysis. The flowchart contained in the Figure 4-7 gives an idea about how the task of the module CIRCLEV is accomplished.

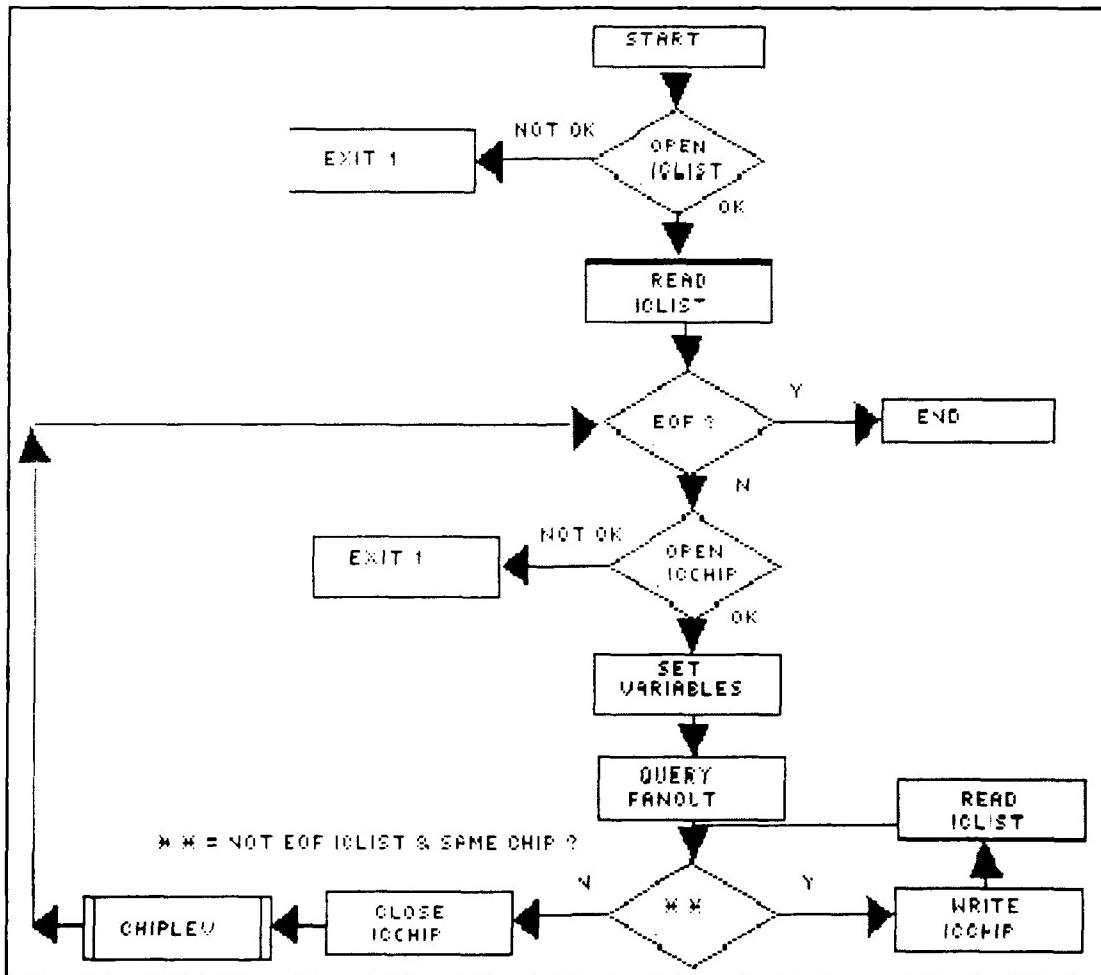


Figure 4-7 CIRCLEV Flowchart

The module CIRCLEV, implemented as a procedure has its construction based on two loops. In the outer loop, attempts are made to open the file ICLIST for reading, and the file ICCHIP for writing. The file ICLIST is opened just once, but the file ICCHIP is opened and closed as many times as there are chips in the circuit. The failure, for any reason, in the opening of either one of those files causes the emission of an error message notifying the fact and immediate abandon of ICE.

Still in the outer loop is the module SET VARIABLES, implemented as a set of statements, which functions to set global variables (TTL type and package identification from the chip currently investigated). Once the kind of TTL under examination is established, an access to the central database is done to determine and retrieve the fanout corresponding to the TTL. This access is made through the module QUERY-FANOUT.

Module QUERY-FANOUT

The module QUERY-FANOUT, implemented as a procedure accesses the relation TTLS in the central database. The access key is the TTL type (e. g. 7408), passed to the procedure as an argument. The procedure returns an integer which indicates the fanout attributed to the type of TTL passed as an argument.

The inner loop is composed of a module that reads ICLIST, and another that writes the file ICCHIP. The inner loop is traversed while the EOF of ICLIST is not reached and the TTL type constant in the record recuperated from ICLIST is the same as the one placed by SET VARIABLES and used as TTL type comparison. If a different TTL type is read from ICLIST and is not EOF of ICLIST, the module CHIPLEV, last part of the

outer loop, is executed and after the execution the process restart having as TTL type comparison the last different accessed. If EOF is reached the module CHIPLEV is performed and the module CIRCLEV finishes its execution.

Module CHIPLEV

In this module each chip is decomposed into gates for an detailed analysis. The flowchart in the Figure 4-8 diagrams the tasks performed in the module CHIPLEV.

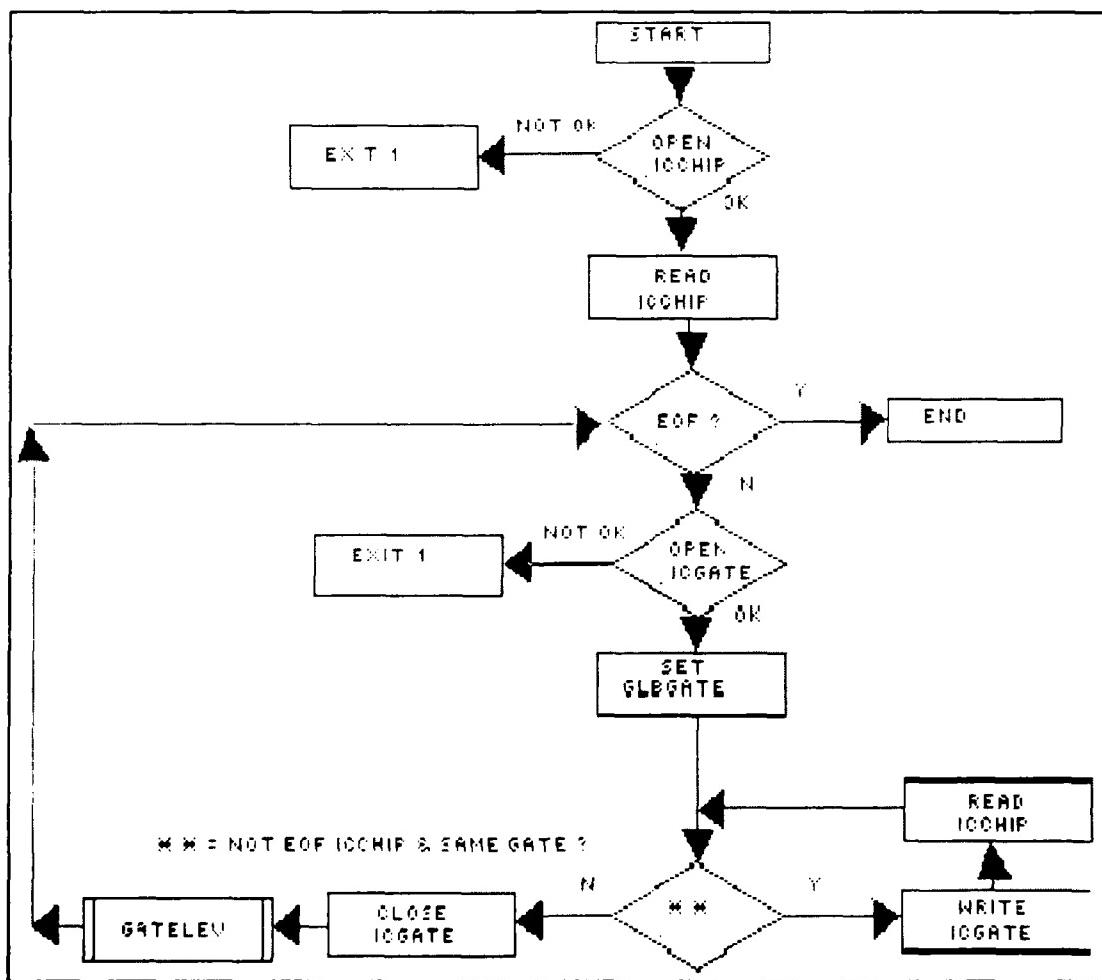


Figure 4-8 CHIPLEV Flowchart

The module CHIPLEV, implemented as a procedure, is very similar to the module CIRLEV. This module also has its construction based on two loops. In the outer loop attempts are made to opening the file ICCHIP for reading and the file ICGATE for writing. Each version of the file ICCHIP implies as many versions of the file ICGATE as the number of gates ,of that particular chip, utilized in the circuit. If for any reason it is impossible to open ICCHIP and ICGATE, the ICE system is abandoned with first happening an generation of an error message corresponding to the problem encountered.

The module SET GLBGATE, still part of the outer loop and implemented as a set of statements, sets the global variable GLBGATE that is used for comparison in the rest of the procedure.

The inner loop performs reading in the file ICCHIP and writing in the file ICGATE while the EOF of ICCHIP is not reached and the type of gate found in the record retrieved from ICCHIP is the same as the one placed by SET GLBGATE.

If a different gate is read from ICCHIP and it is not EOF of ICCHIP the module GATELEV, last part of the outer loop, is executed. After GATELEV execution the process is reactivated having as GLBGATE the last different gate accessed.

If EOF of ICCHIP is found, the module GATELEV still is performed, but after its execution, the module CHIPLEV finishes its performance passing the control to the calling procedure CIRCLEV.

Module GATELEV

At this module, implemented as a procedure, is made the analysis to find out about the existence of missing links. The Figure 4-9 depicts how the tasks are performed in the module.

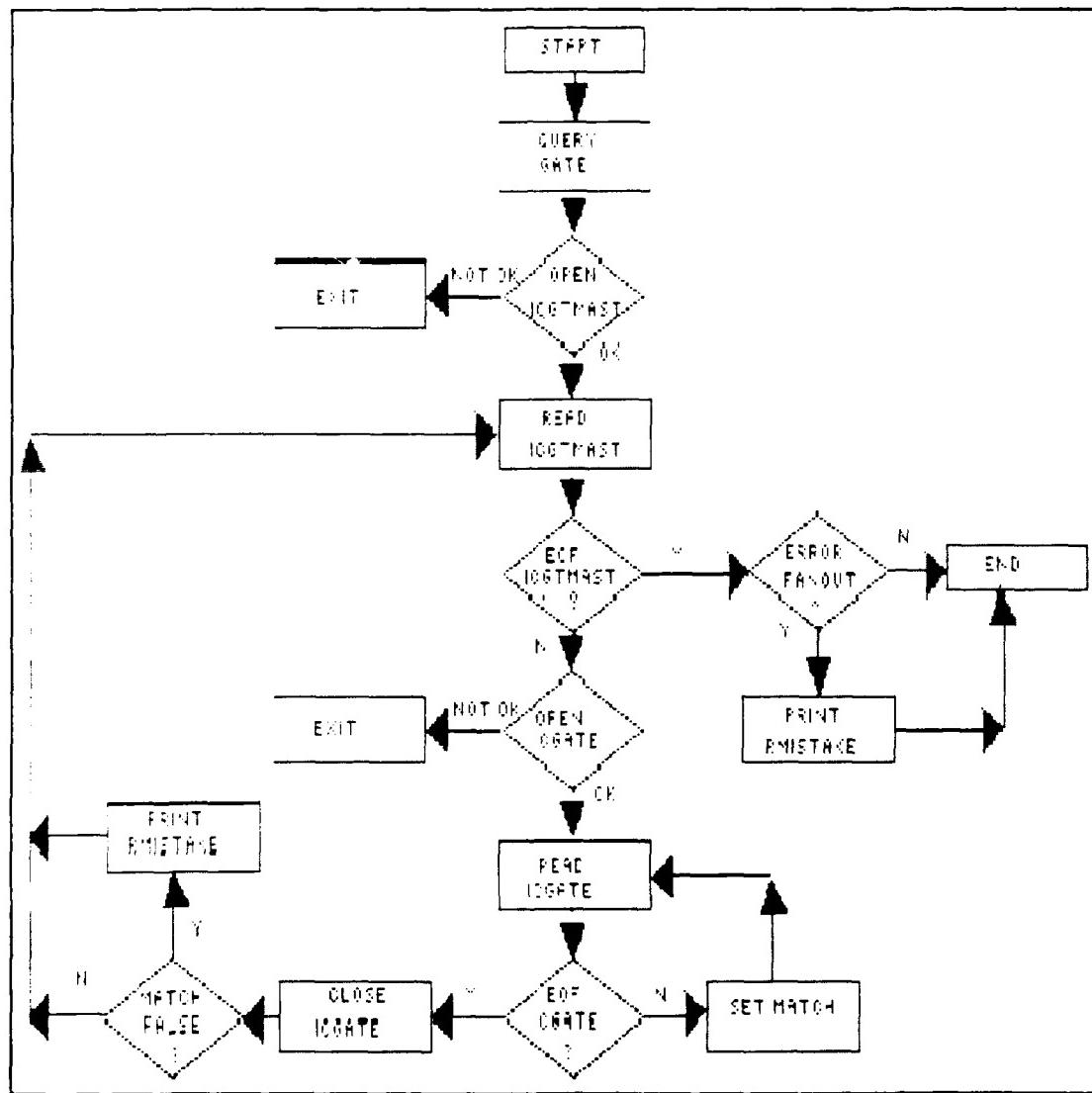


Figure 4-9 GATELEV Flowchart

The module GATELEV starts making an access to the central database through the module QUERY-GATE.

Module QUERY-GATE

This module, implemented as a procedure, accesses the relation PINS in the central database for retrieving all information related to a specific gate of a specific TTL. This module returns a file called ICGTMAST containing all information retrieved from the central database which means that each record in the file has pin number associated to the gate, gate name (e. g A,B,C, etc), and gate value (e.g INPUT, OUTPUT, CLOCK, GROUND, POWER, and NC meaning that this particular pin should not be connected).

After completion of the QUERY-GATE, attempts to open ICGTMAST and ICGATE are made. If the attempts fail, error messages are issued to the user and ICE ends its execution. If the attempt succeeds, the execution of GATELEV continues setting the matching between the information contained in the files ICGTMAST and ICGATE. This match setting try, represented by the module SET MATCH is made until EOF of ICGATE is reached or a True match is done. The value of the match is set to False, if the information existing in the files are non consistent, or in other words a port of a gate exists in the master gate file (ICGTMAST) but the respective connection to this port of the gate is not present in the circuit. After the completion of SET MATCH the match value is checked. If the match value is False, the module PRINT-RMISTAKE is executed.

Module PRINT-RMISTAKE

This module, implemented as a procedure has as its basic function to place, in the file RMISTAKE, all messages concerned with missing links, as well as the messages concerned with fanout errors. The handling of fanout errors message is explained later in this chapter.

After the execution of the match test clause, with accomplishment or not of the module PRINT-MISTAKE, another record from ICGTMAST is read, and the cycle is restarted. The cycle is executed until EOF of ICGTMAST is found, then a test is performed to find out if the links to the gate do not violate the limits imposed by the fanout. If there is violation of the fanout the module PRINT-RMISTAKE is executed in order to put the corresponding message into RMISTAKE, after that GATELEV is finished and the control is passed to the calling procedure CHIPLEV. If there is not a fanout violation GATELEV finishes and the calling procedure, CHIPLEV, assumes the control of the ICE system.

Module WRAPUP

This module is responsible for constructing the output file called ICEOUT.TXT. As described earlier in the section OUTPUT PHASE of the chapter 3, there are four possible manners to accomplish the ICEOUT.TXT.

1 - YMISTAKE and RMISTAKE exist. In this case Ymistake is renamed ICEOUT.TXT and RMISTAKE is appended to it;

2 - Just Ymistake exists. This case requires only the renaming of the file YMISTAKE to ICEOUT.TXT;

3 - Only Rmistake exists. Here, Rmistake is renamed ICEOUT.TXT;

4 - There is no YMISTAKE, or RMISTAKE. If this case happens, ICEOUT.TXT is created having as its unique record the sentence "No output from ICE", what means that ICE was executed and no error was flagged in the circuit.

The description of the module WRAPUP closes this chapter.

V - Testing and Results

This chapter shows the criteria utilized in the testing phase as well as the results obtained.

In order to evaluate the New ICE, the conducted tests aimed to inspect either the functional capability and the efficiency of the new system.

Functionality Tests

Several tests cases were developed to prove that each rule could be "fired". The tests covered the cases of no errors in the circuit, one case of each possible error, and also diverse combinations of errors. The system always reached the expected results. During this phase, it has been made a monitoring in each module to assure their correct functioning.

Integration

The integration of ICE to the central database, logsim, and graphics interface was tested, and the test succeeded.

Efficiency Tests

The efficiency tests were conducted to see if the requirements established in the design phase and in the problem statement were satisfied. A description of the tests follows.

Portability

The portability exists with the new ICE. The system was installed in several types of computer (Zenith, IBM PCAT, IBM clone) and ran without any

modification. The outcomes from the system in all environments were the same.

Independence of Commercial Package

A complete independence from any commercial package was reached. The new ICE exists by itself. Unlike the old system that relies on GURU, the new system does not need any tool as support.

Response Time

The response time has been measured from the starting of the execution of ICE until first moment when the output file was made available to the user. The response time of the new ICE, in all occasions, was better than the response time obtained with the old system analyzing the same circuit. The goal of reducing the response time to half was achieved. In the section RESULTS are shown graphics with the response time tests outcomes.

Space requirements

The amount of external memory (disk) required for the new ICE is much less than the space required by the old system. The old system requires, for its 54 files, nearly 1600 Kbytes of disk space. The new system, with 3 files, requires nearly 125 Kbytes. The space required on main memory also favors the new system, since it can run with other resident system (e.g. Norton Commander), what is totally impossible by the old system that allocates all the memory.

Results

Response Times

The tests were conducted running identical circuits under the NEW ICE and the OLD ICE. The machine at the systems were tested has been a personal computer Zenith 248 with the following characteristics:

- . base memory size - 640 kbytes,
- . expansion memory - 2560 kbytes,
- . video display enhanced graphics,
- . video refreshing rate 60 Hz,
- . hard drive 21 Mbytes of capacity.

The results obtained were put into the graph shown in Figure 5-1. The legends are explained in the following way: the first part means the system were the circuit was tested and the second part says how many gates per TTL were used in the circuit. So, NEW 4GATES is the legend of curve resulted from running circuits under the new system with each TTL of the circuit utilizing 4 gates.

PERFORMANCE NEW vs OLD SYSTEM

(TTLS vs Time)

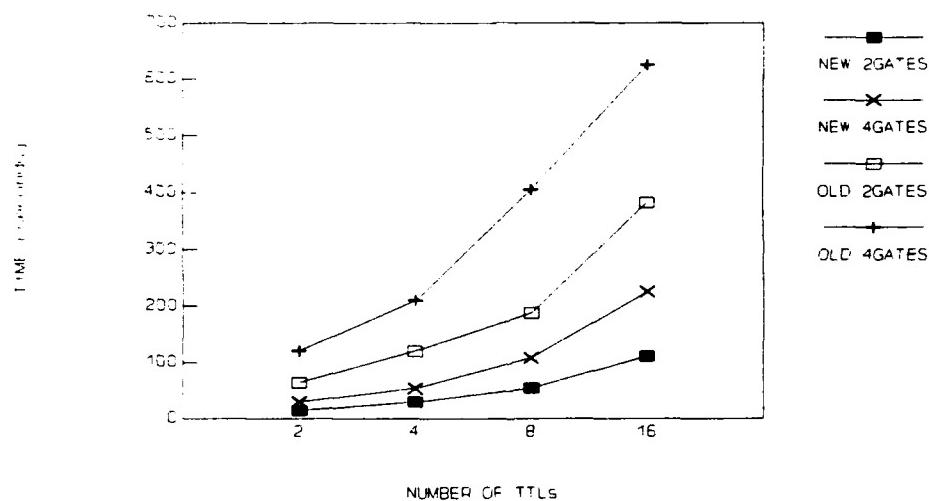


Figure 5-1 Performance New ICE vs Old ICE

The following table shows that the goal of reducing the response time to half of time required by the old system was totally achieved.

CKT TYPE Denomination	Response Time in seconds	
	Under New System	Under Old System
CKT022c	14	64
CKT022i	15	63
CKT024c	30	125
CKT024i	31	124
CKT042c	30	120
CKT042i	31	119
CKT044c	54	204
CKT044i	55	210
CKT082c	54	188
CKT082i	54	195
CKT084c	108	405
CKT084i	110	410
CKT162c	110	382
CKT162i	111	390
CKT164c	225	650
CKT164i	228	654

The denomination of circuits types were obtained applying the following name formation rule, CKTXXZY, where:

CKT - identifies a digital circuit,

XX - the two first digits shows how many TTLs were used in the circuit,

Z - how many gates per TTL were used, and

Y - assuming "c" for correct and "i" for incorrect circuits.

The overall response time performance of the new ICE seems to obey a linear function of the number of gates. As the number of the gates increases, there is a linear increase in time. Figure 5-2 illustrates the linear feature of the new ICE.

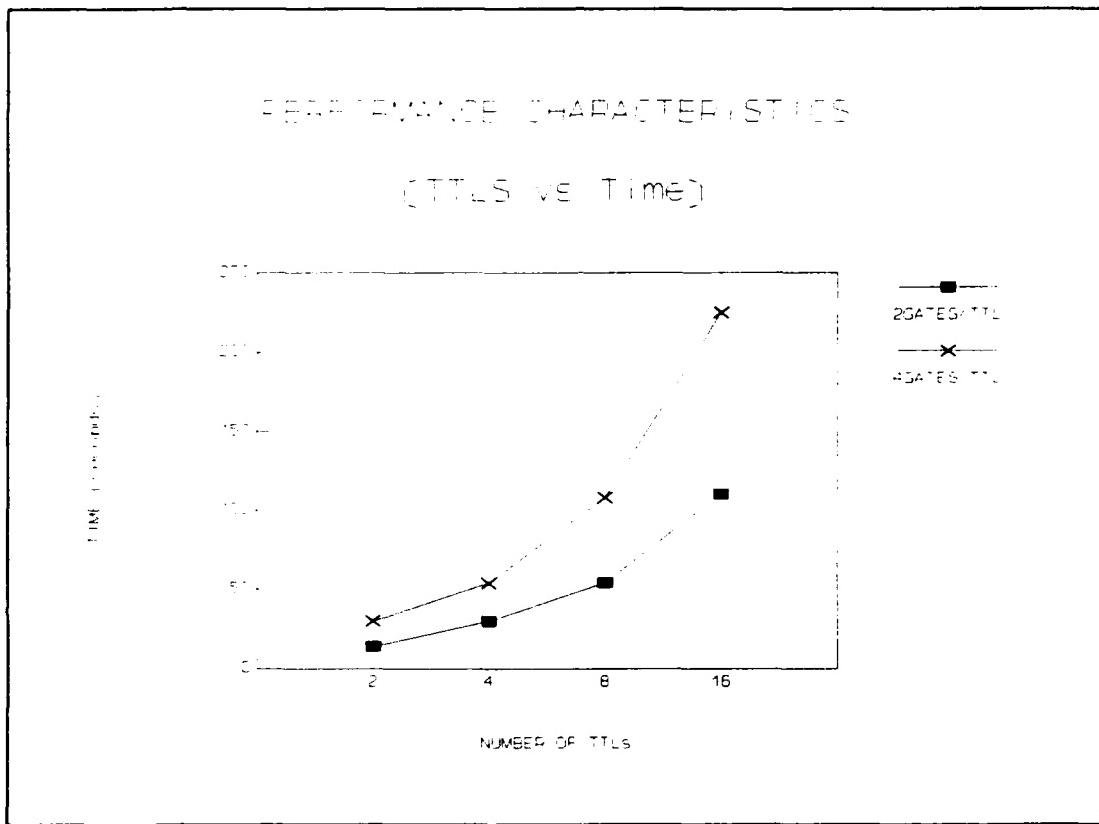


Figure 5-2 Performance Characteristic of the New ICE (Time vs TTLs)

From this graphs and from the previous table it is possible infer that although, it highly desired and recommended to use all gates of a TTL before put a new TTL in the circuit, the new ICE does not show a significant difference in response time between a circuit that uses 32 gates either using 8 chips with 4 gates on each chip or 16 chips with 2 gates on each chip. This reenforces the idea that the system is more sensible to the number of gates than to the number of TTLs.

Follow some graphs showing results obtained from new ICE when after testing correct and incorrect circuits with variable number of gates. Wrong means the time spent finding the wrong connections, MISSING the time spent finding missing connections, and I/O the time spent with input and output.

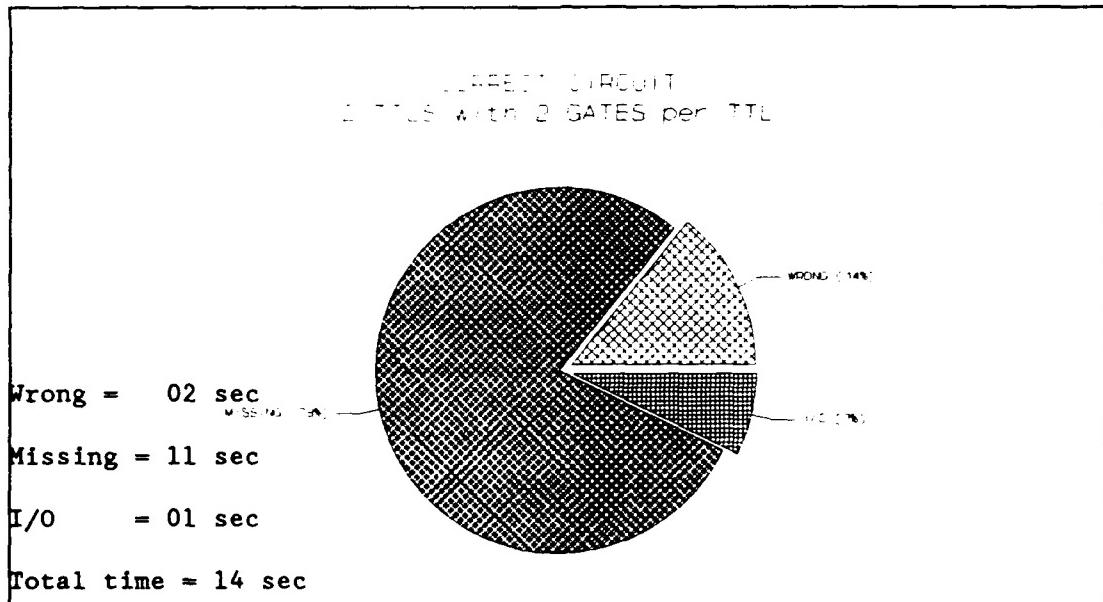


Figure 5-3 Correct Circuit with 2 TTLs / 2 Gates per TTL

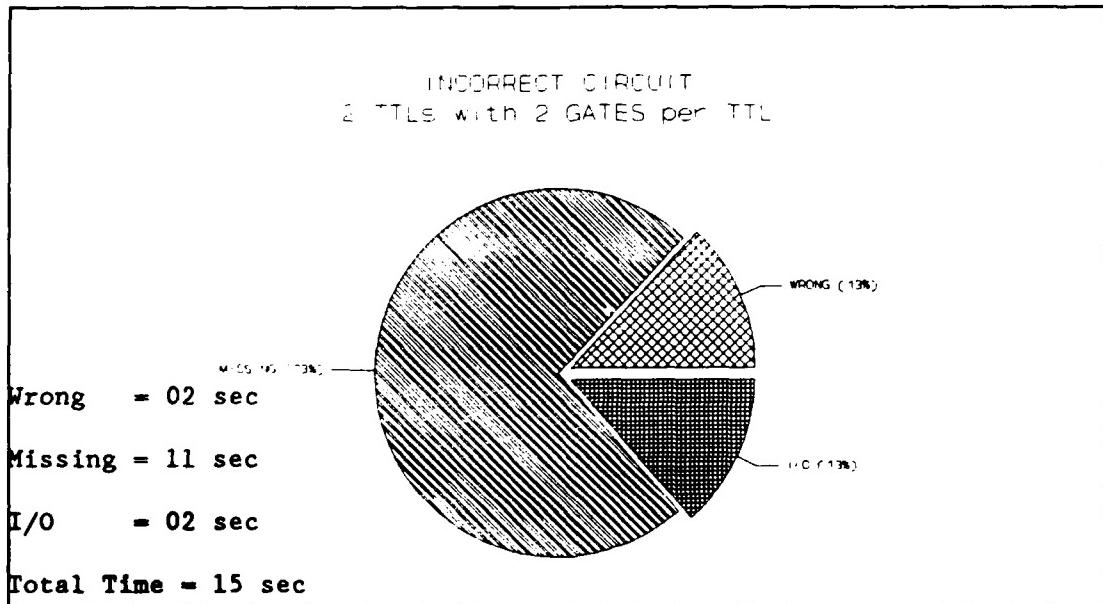


Figure 5-4 Incorrect Circuit with 2 TTLs / 2 Gates per TTL

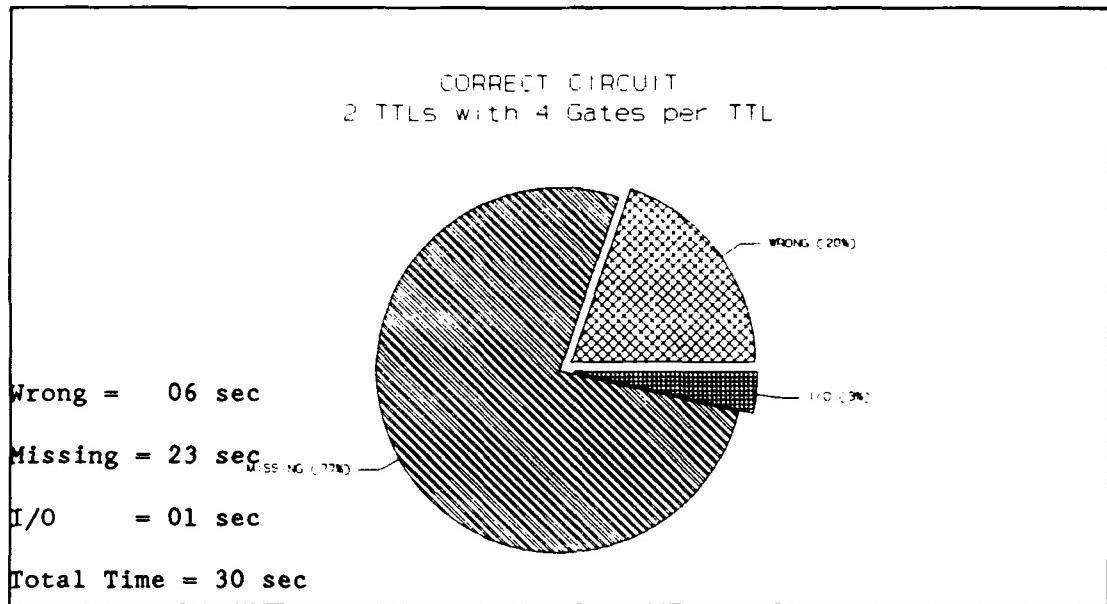


Figure 5-5 Correct Circuit with 2 TTLs / 4 Gates each TTL

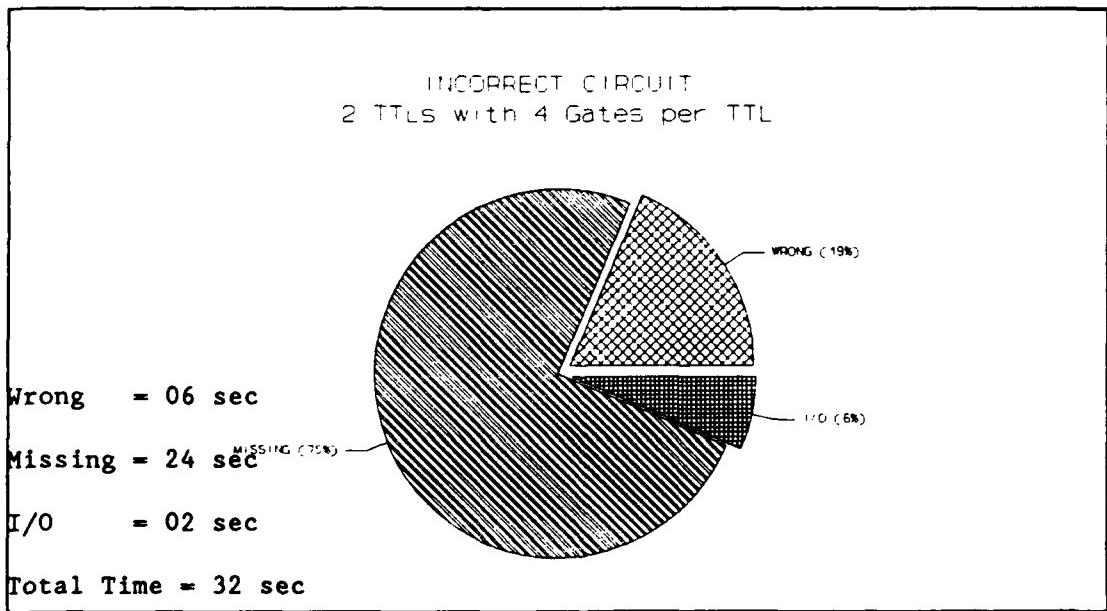


Figure 5-6 Incorrect Circuit with 2 TTLs / 4 Gates per TTL

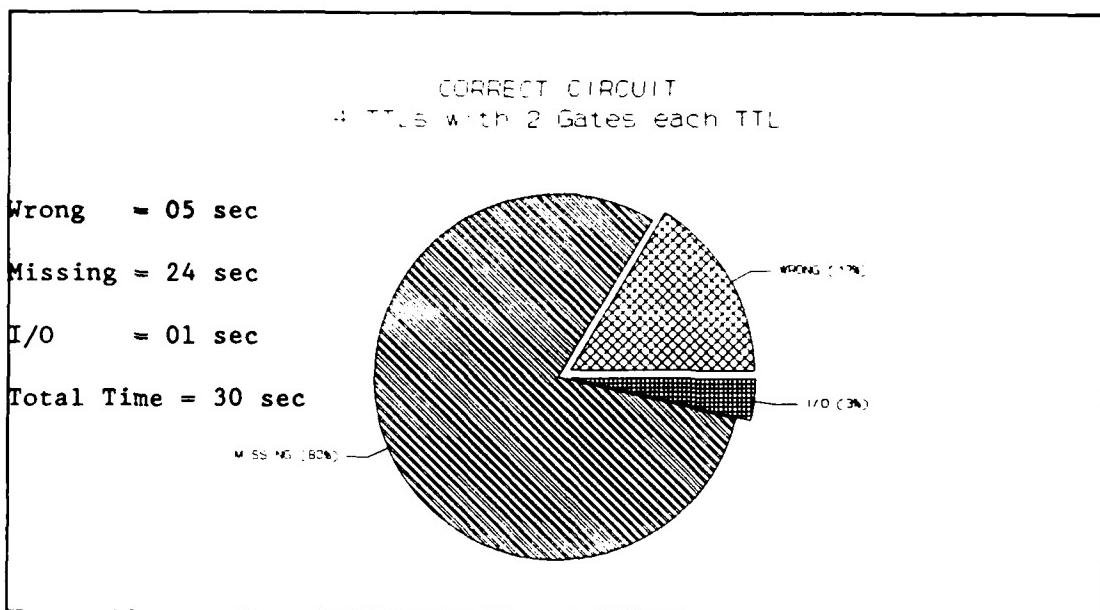


Figure 5-7 Correct Circuit with 4 TTLs / 2 Gates per TTL

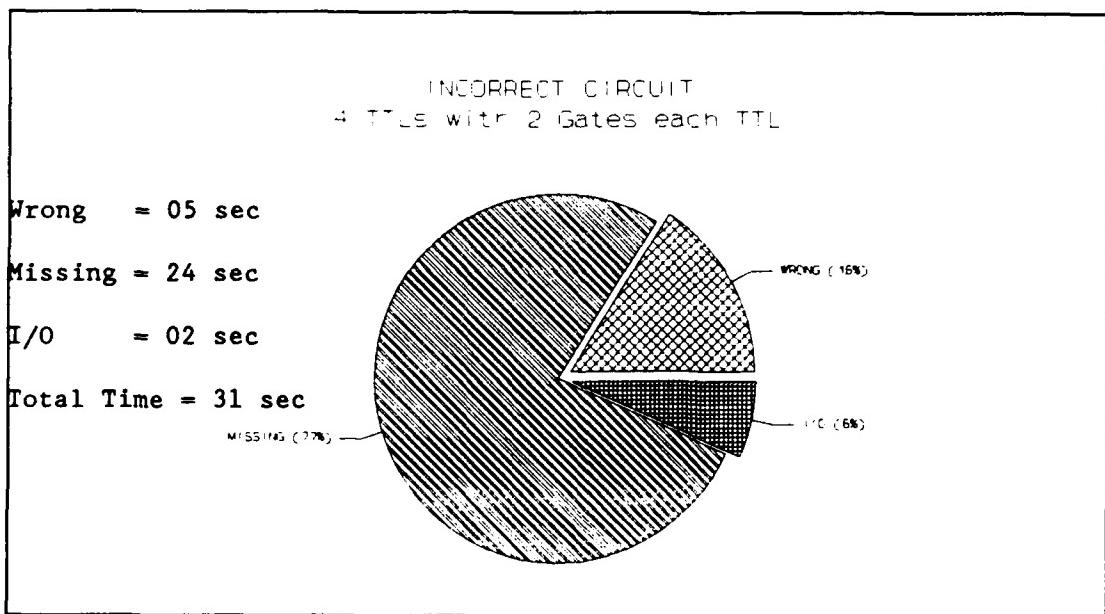


Figure 5-8 Incorrect Circuit with 4 TTLs / 2 Gates per TTL

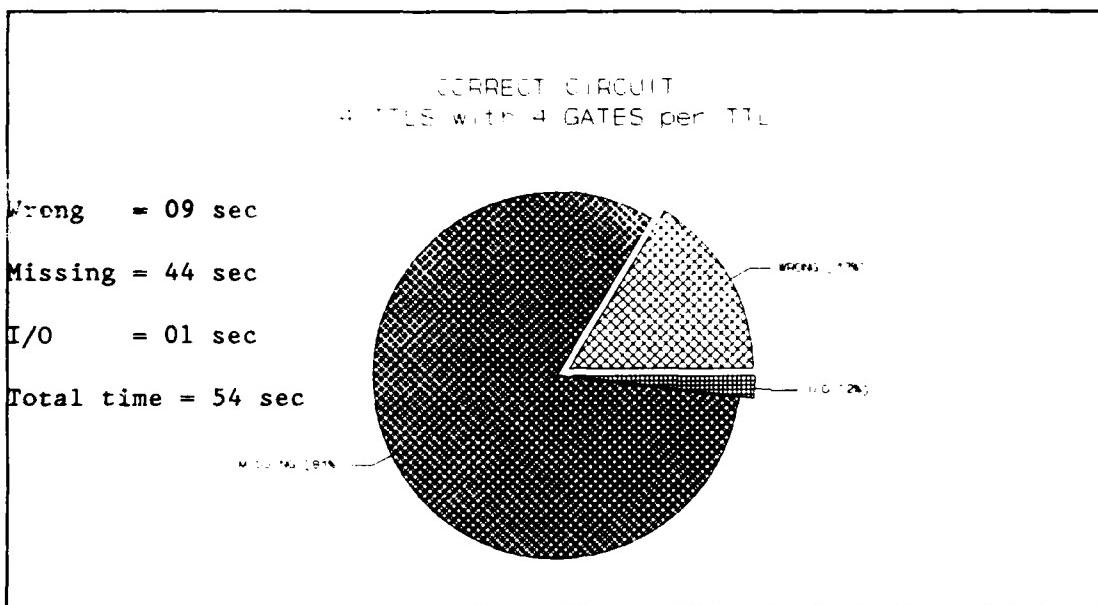


Figure 5-9 Correct Circuit with 4 TTLs / 4 Gates per TTL

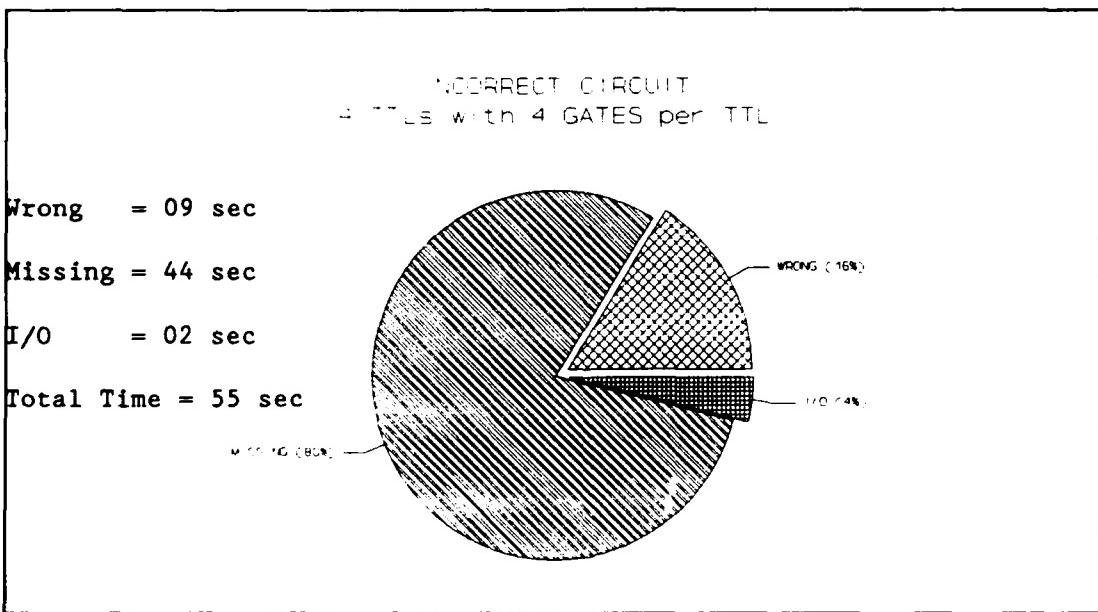


Figure 5-10 Incorrect Circuit with 4 TTLs / 4 Gates per TTL

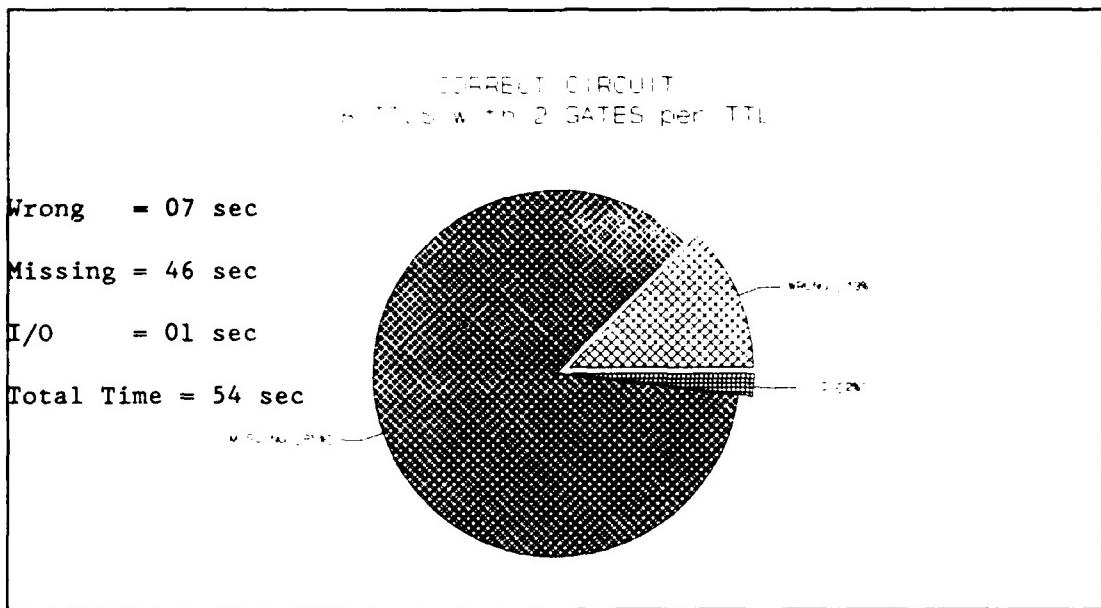


Figure 5-11 Correct Circuit with 8 TTLs / 2 Gates per TTL

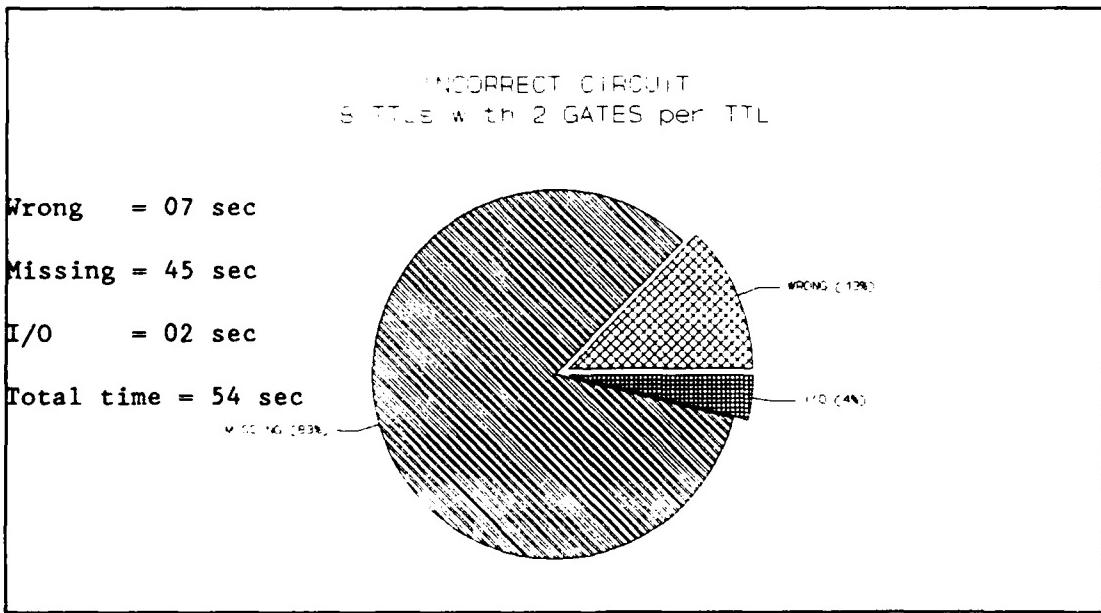


Figure 5-12 Incorrect Circuit with 8 TTLs / 2 Gates per TTL

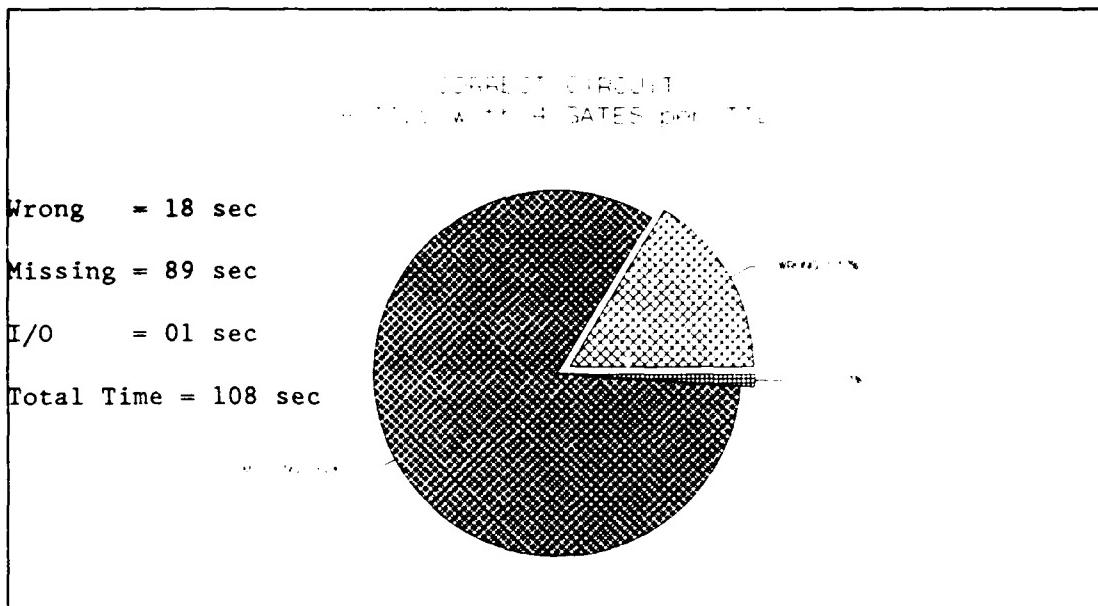


Figure 5-13 Correct Circuit with 8 TTLs / 4 Gates per TTL

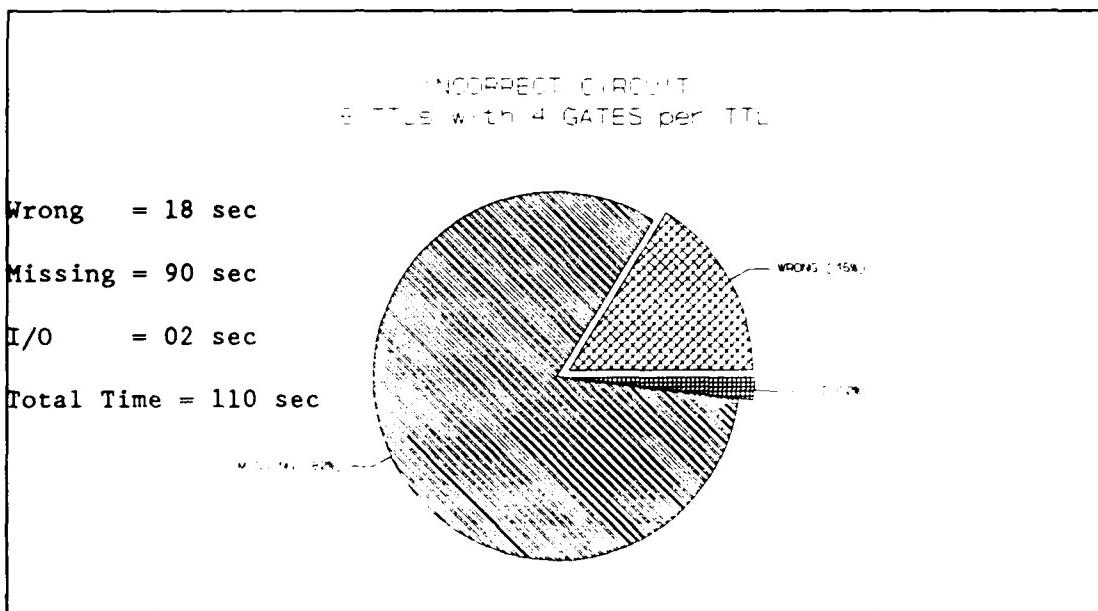


Figure 5-14 Incorrect circuit with 8 TTLs / 4 Gates per TTL

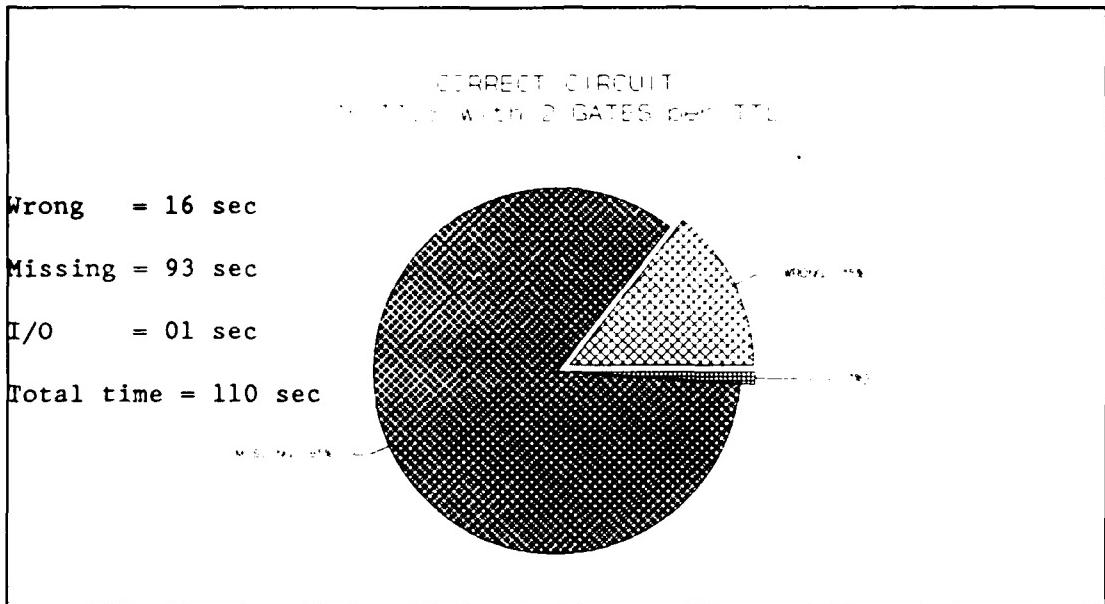


Figure 5-15 Correct Circuit with 16 TTLs / 2 Gates per TTL

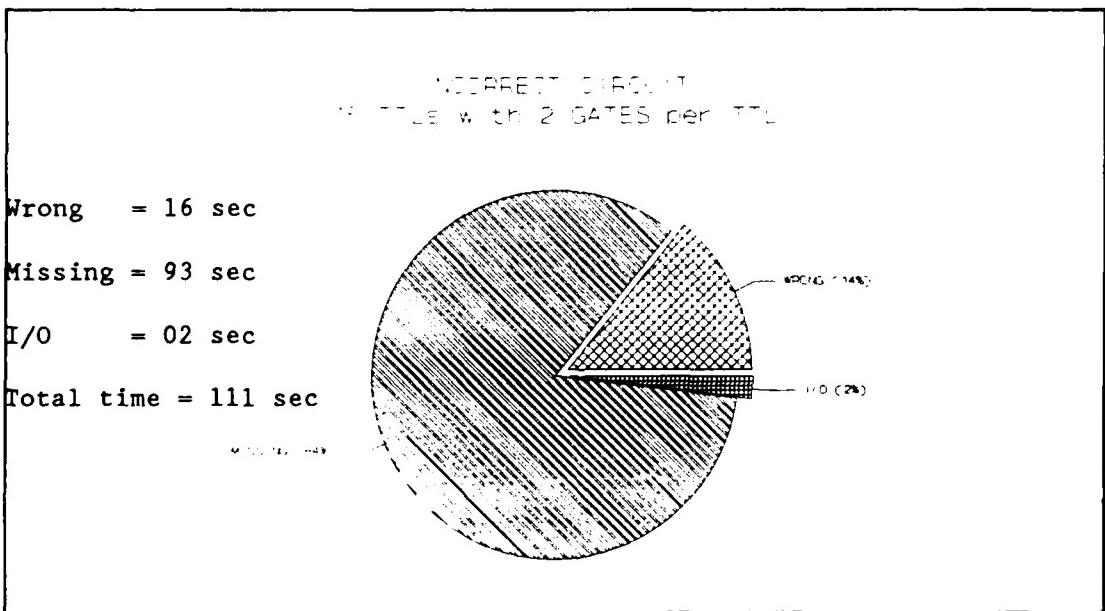


Figure 5-16 Incorrect Circuit with 8 TTLs / 2 Gates per TTL

The percentage of time spent on finding missing connections is consistently of the total time spent to analyze the entire circuit. There are several reasons for the long delay in finding the missing connections. One of the causes for the delay is the algorithm used that checks the connections individually. Another reason is the time spent read/writing to the disk. The new ICE uses intermediate file extensively during the processing phase.

Space Requirements

The relation between the occupied space in disk by both systems can be seen in Figure 5-11. The implementation of the new ICE brought a reduction of an order of 93 % in the size of the old system.

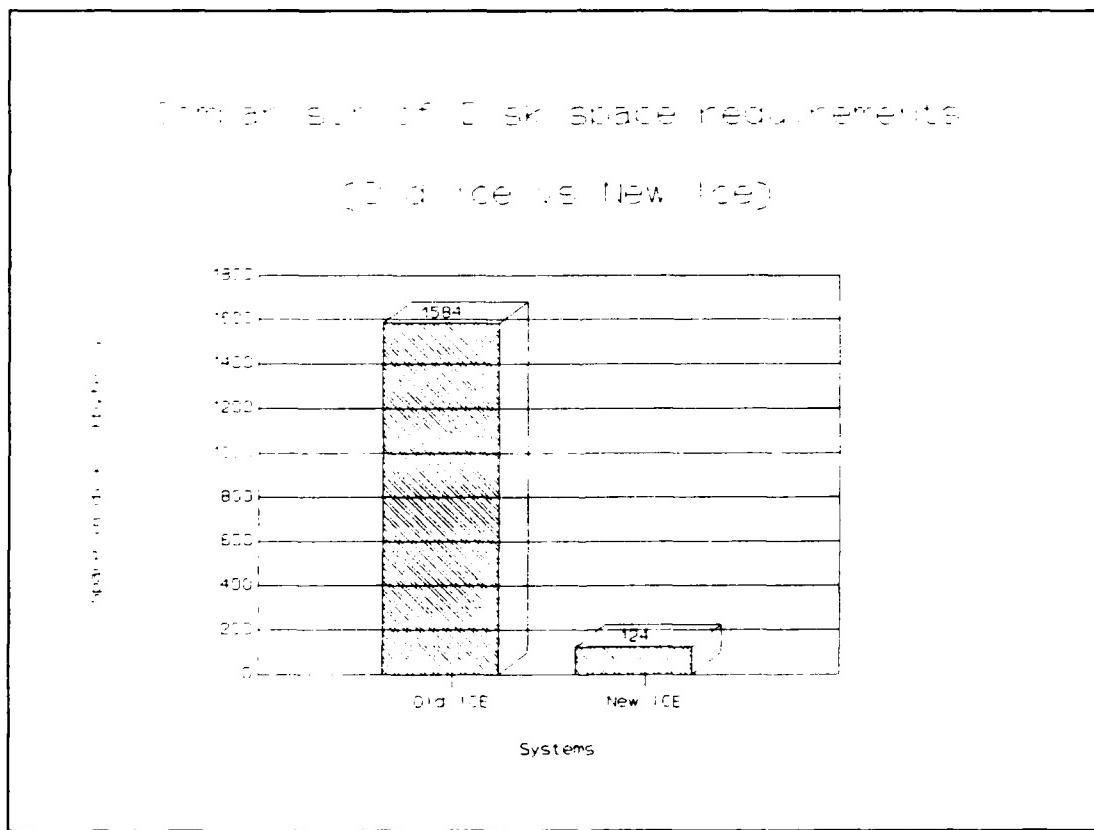


Figure 5-17 Comparison of Disk Space Requirements (Old ICE vs New ICE)

Summary

The new ICE which resulted from this thesis effort satisfied all the requirements formulated. The space on disk was reduced, the amount of memory also was reduced, there is no longer any dependency of a commercial package, the response time was reduced to less than half time of the old system, the new ice is portable, and an integration with the central database, LOGSIM, and Graphics Interface was established.

VI - Conclusions and Recommendations

In this chapter, the main research findings related to design and implementation of ICE are summarized. Recommendations for further research are indicated.

Conclusions

The main research findings are summarized as follow:

1 - Although the programming language "C" was not design specially designed to deal with problems in the AI field, it can be used with success.

2 - The technique "Divide and Conquer" or the Descartes' Principles of Analysis and Synthesis", where a complex problem is decomposed in small solvable subproblems, once more shown be applicable and efficient.

3 - A well done plan is essential for the success of a task. With the plan is possible to solve all interfaces problems when integration with other parts is necessary. During the effort of this thesis many problems did not arise because was planned interactions with the people responsible for developing the other integrating parts of the system. A good example was the utilization of the database by ICE. Meetings held with Captain Sue A. Ehrhart resulted in benefits either to me, because I found out the database potential, as well as to Captain Ehrhart that became aware of the necessities of the database user.

4 - Prototyping demonstrated be a very useful technique, because by means of the prototype utilization bugs or situations not predicted in the design phase are met, and corrected with a low cost for the entire project.

5 - General programs or packages are good but they bring an extra cost to perform simple tasks. The case GURU-ICE is typical, GURU is a powerful tool, it has many features but just for that it requires a great deal of memory and the processing time required to perform simple tasks increases. The relation cost/benefit in the case proved not be favorable to GURU. The developed ICE is limited in its scope but solve the AFIT problem with much more property.

Recommendations

The speed in the new ICE was increased but it certainly would be better if instead checking the entire circuit it could check an individual chip. Also, utilization of a virtual disk, ram disk, would minimize the time spent in the process of finding missing connections which represents, in average, 80% of the total time spent by ICE during the analysis of a digital circuit.

In this way it is recommended to make researches in order to provide ICE with a "mechanism" that could detect the existence of a ramdisk which would be used as scratch pad for the intermediate files necessary for the ICE execution.

Appendix A: ICE Rules Set

Rule : R1

```
IF : (iceta.fgate = "NC") or (iceta.tgate = "NC")
THEN : (iceta.clor <- error)
```

Rule : R2

```
IF : (iceta.fpkg[0] = iceta.tpkg[0])
THEN : (same <- TRUE)
```

Rule : R3

```
IF : (iceta.fpkg = iceta.tpkg)
THEN : (samechip <- TRUE)
```

Rule : R4

```
IF : (iceta.fgate = iceta.tgate)
THEN : (samegate <- TRUE)
```

Rule : R5

```
IF : (iceta.fpinval = iceta.tpinval)
THEN : (sameval <- TRUE)
```

Rule : R6

```
IF : (((iceta.fpinval = "INPUT") and (iceta.tpinval = "OUTPUT")) or  
      ((iceta.fpinval = "OUTPUT") and (iceta.tpinval = "INPUT")) and  
      (not (samechip and samegate)))  
THEN : (iceta.clor <- OK)
```

Rule : R7

```
IF : (same and samechip and samegate)  
THEN : ((iceta.clor <- ERROR)  
        (print in Ymistake " y package iceta.fpkg has a race condition  
present between pins iceta.fpin and iceta.tpin"))
```

Rule : R8

```
IF : (same and sameval)  
THEN : ((iceta.clor <- ERROR)  
        (print in Ymistake "y Two iceta.fpinval are tied together pin  
iceta.fpin of iceta.fpkg and iceta.tpin of iceta.tpkg"))
```

Rule : R9

```
IF : (iceta.fpinval = POWER) and (iceta.tpinval in (INPUT, CLOCK,  
GROUND, OUTPUT))  
THEN : ((iceta.clor <- ERROR)  
        (print in Ymistake "y Pin iceta.fpin, a iceta.fpinval of  
iceta.fpkg is incorrectly tied to iceta.tpkg, pin iceta.tpin a  
iceta.tpinval))
```

Rule : 10

```
IF : (iceta.fpinval = GROUND) and (iceta.tpinval in (INPUT, CLOCK,  
POWER, OUTPUT))  
  
THEN : ((iceta.clor <- ERROR)  
  
(print in Ymistake "y Pin iceta.fpin, a iceta.fpinval of  
iceta.fpkg is incorrectly tied to iceta.tpkg, pin iceta.tpin a  
iceta.tpinval))
```

Rule : R11

```
IF : (iceta.fpinval = CLOCK) and (iceta.tpinval in (INPUT, POWER,  
GROUND, OUTPUT))  
  
THEN : ((iceta.clor <- ERROR)  
  
(print in Ymistake "y Pin iceta.fpin, a iceta.fpinval of  
iceta.fpkg is incorrectly tied to iceta.tpkg, pin iceta.tpin a  
iceta.tpinval))
```

Rule : R12

```
IF : (iceta.fpinval = INPUT) and (iceta.tpinval in (POWER, CLOCK,  
GROUND, OUTPUT))  
  
THEN : ((iceta.clor <- ERROR)  
  
(print in Ymistake "y Pin iceta.fpin, a iceta.fpinval of  
iceta.fpkg is incorrectly tied to iceta.tpkg, pin iceta.tpin a  
iceta.tpinval))
```

Rule : R13

IF : (iceta.fpinval = OUTPUT) and (iceta.tpinval in (INPUT, CLOCK,
GROUND, POWER))

THEN : ((iceta.clor <- ERROR)

(print in Ymistake "y Pin iceta.fpin, a iceta.fpinval of
iceta.fpkg is incorrectly tied to iceta.tpkg, pin iceta.tpin a
iceta.tpinval))

Rule : R14

IF : (not (iceta.fpkg[0] = iceta.tpkg[0]))

THEN : (notsame <- TRUE)

Rule : R15

IF : ((notsame) and (iceta.fpkg = TTL) and (iceta.fpinval = CLOCK) and
(iceta.tpinval in (INPUT, CLOCK)))

THEN : (iceta.clor <- OK)

Rule : R16

IF : ((notsame) and (iceta.fpinval in (INPUT,CLOCK)) and (iceta.tpkg =
TTL) and (iceta.tpinval = CLOCK))

THEN : (iceta.clor <- OK)

Rule : R17

IF : ((notsame) and (iceta.fpkg = TTL) and (iceta.fpinval = GROUND) and
(iceta.tpinval in (INPUT, GROUND)))

THEN : (iceta.clor <- OK)

Rule : R18

IF : ((notsame) and (iceta.fpinval in (INPUT, GROUND)) and (iceta.tpkg = TTL) and (iceta.tpinval = GROUND))
THEN : (iceta.clor <- OK)

Rule : R19

IF : ((notsame) and (iceta.fpkg = TTL) and (iceta.fpinval = POWER) and (iceta.tpinval in (INPUT, POWER)))
THEN : (iceta.clor <- OK)

Rule : R20

IF : ((notsame) and (iceta.fpkg = TTL) and (iceta.fpinval = INPUT) and (iceta.tpinval in (INPUT, GROUND, POWER)))
THEN : (iceta.clor <- OK)

Rule : R21

IF : ((notsame) and (iceta.fpinval in (INPUT, POWER)) and (iceta.tpkg = TTL) and (iceta.tpinval = POWER))
THEN : (iceta.clor <- OK)

Rule : R22

IF : ((notsame) and (iceta.fpinval in (INPUT, GROUND, POWER)) and (iceta.tpkg = TTL) and (iceta.tpinval = INPUT))
THEN : (iceta.clor <- OK)

Rule : R23

```
IF : ((notsame) and (iceta.fpkg = TTL) and (iceta.fpinval = OUTPUT) and  
(iceta.tpinval in (INPUT, POWER, GROUND, CLOCK)))  
  
THEN : ((iceta.clor <- ERROR)  
  
(print in Ymistake "y external iceta.tpinval is mistakenly  
connected to a iceta.fpinval pin of iceta.fpkg, pin iceta.fpin"))
```

Rule : R24

```
IF : ((notsame) and (iceta.fpkg = TTL) and (iceta.fpinval = CLOCK) and  
(iceta.tpinval in (POWER, GROUND)))  
  
THEN : ((iceta.clor <- ERROR)  
  
(print in Ymistake "y external iceta.tpinval is mistakenly  
connected to a iceta.fpinval pin of iceta.fpkg, pin iceta.fpin"))
```

Rule : R25

```
IF : ((notsame) and (iceta.fpkg = TTL) and (iceta.fpinval = GROUND) and  
(iceta.tpinval in (POWER, CLOCK)))  
  
THEN : ((iceta.clor <- ERROR)  
  
(print in Ymistake "y external iceta.tpinval is mistakenly  
connected to a iceta.fpinval pin of iceta.fpkg, pin iceta.fpin"))
```

Rule : R26

```
IF : ((notsame) and (iceta.fpkg = TTL) and (iceta.fpinval = POWER) and  
(iceta.tpinval in (GROUND, CLOCK)))  
  
THEN : ((iceta.clor <- ERROR)  
  
(print in Ymistake "y external iceta.tpinval is mistakenly  
connected to a iceta.fpinval pin of iceta.fpkg, pin iceta.fpin"))
```

Rule : R27

```
IF : ((notsame) and (iceta.fpkg = TTL) and (iceta.fpinval = INPUT) and  
(iceta.tpinval = CLOCK))  
  
THEN : ((iceta.clor <- ERROR)  
  
(print in Ymistake "y external iceta.tpinval is mistakenly  
connected to a iceta.fpinval pin of iceta.fpkg, pin iceta.fpin"))
```

Rule : R28

```
IF : ((notsame) and (iceta.fpinval in (INPUT, POWER, GROUND, CLOCK)) and  
(iceta.tpkg[0] = TTL) and (iceta.tpinval = OUTPUT))  
  
THEN : ((iceta.clor <- ERROR)  
  
(print in Ymistake "y external iceta.fpinval is mistakenly  
connected to a iceta.tpinval pin of iceta.tpkg, pin iceta.tpin"))
```

Rule : R29

```
IF : ((notsame) and (iceta.fpinval in (POWER, GROUND)) and  
(iceta.tpkg[0] = TTL) and (iceta.tpinval = CLOCK))  
  
THEN : ((iceta.clor <- ERROR)  
  
(print in Ymistake "y external iceta.fpinval is mistakenly  
connected to a iceta.tpinval pin of iceta.tpkg, pin iceta.tpin"))
```

Rule : R30

```
IF : ((notsame) and (iceta.fpinval in (POWER, CLOCK)) and (iceta.tpkg[0]  
= TTL) and (iceta.tpinval = GROUND))  
  
THEN : ((iceta.clor <- ERROR)  
  
(print in Ymistake "y external iceta.fpinval is mistakenly  
connected to a iceta.tpinval pin of iceta.tpkg, pin iceta.tpin"))
```

Rule : R31

```
IF : ((notsame) and (iceta.fpinval in (GROUND, CLOCK)) and  
(iceta.tpkg[0] = TTL) and (iceta.tpinval = POWER))  
  
THEN : ((iceta.clor <- ERROR)  
  
(print in Ymistake "y external iceta.fpinval is mistakenly  
connected to a iceta.tpinval pin of iceta.tpkg, pin iceta.tpin"))
```

Rule : R32

IF : ((notsame) and (iceta.fpinval = CLOCK) and (iceta.tpkg[0] = TTL)
and (iceta.tpinval = INPUT))

THEN : ((iceta.clor <- ERROR)

(print in Ymistake "y external iceta.fpinval is mistakenly
connected to a iceta.tpinval pin of iceta.tpkg, pin iceta.tpin"))

Appendix B:

User's Guide for ICE

To use any of the ICE menu options, the user must click the left button while pointing to the "ICE" option on the "MAIN MENU" of the GRAPH system (See Figure B-1).

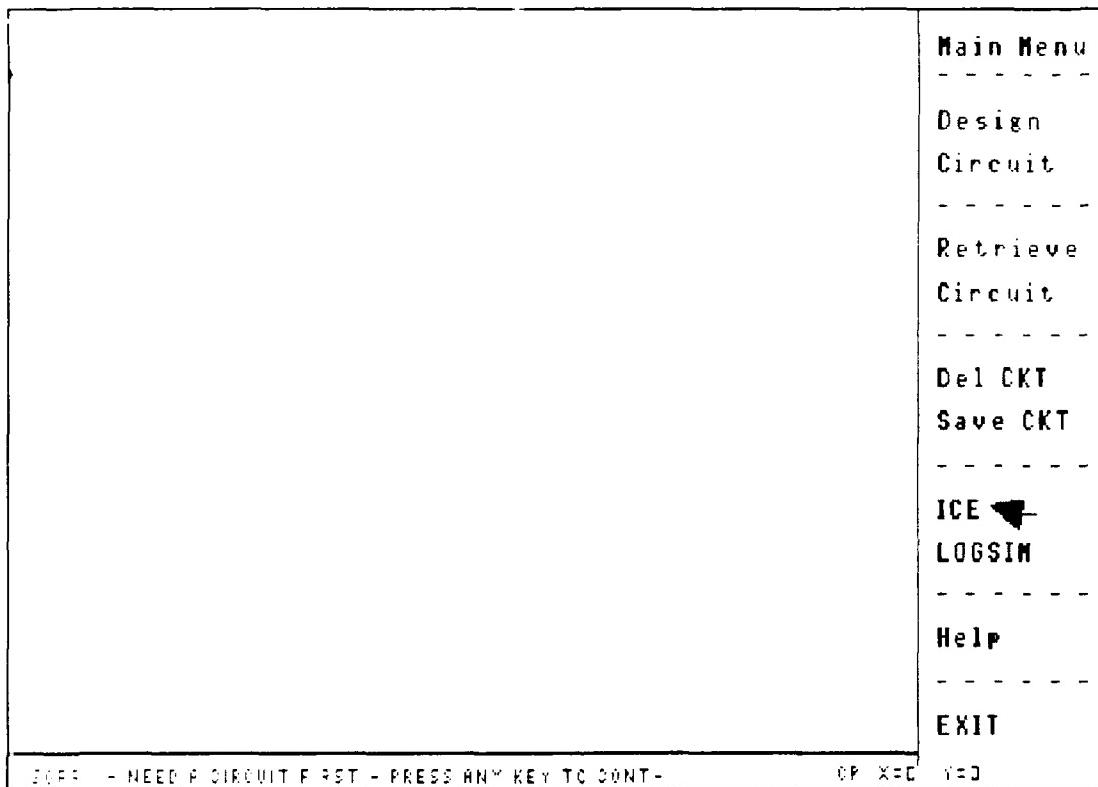


Figure B - 1 Clicking on "ICE" option

If there is no circuit in progress at the time the user will be prompted with : "Sorry - need a circuit first - press any key to cont-" and the ICE title will be highlighted. Once a key is pressed the title will return to normal color. If a circuit is in progress the "ICE Menu" will replace the "Main Menu" (See Figure B-2).

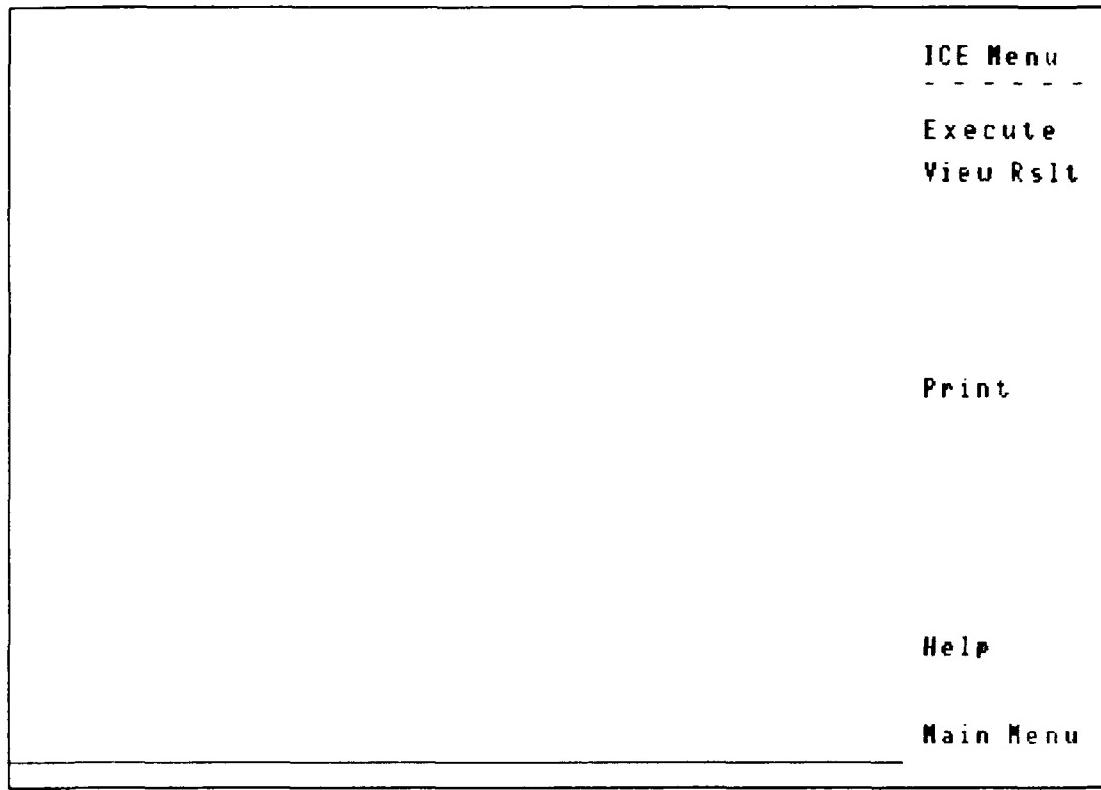


Figure B-2 ICE Menu

Evaluating a circuit

To evaluate a circuit using the ICE program, the user clicks left button while pointing to the "Execute" option in the "ICE Menu". The "Execute" title will remain highlighted until this option is terminated. The user will be advised : "Currently executing ICE - please standby". When ICE has finished, the "Execute" title returns to normal color, and the sentence "Currently executing ICE - please standby", disappears.

Viewing ICE results

To view the output results from ICE the user clicks the left button while pointing to the "View Rslt" option in the "ICE Menu". The current screen display will be replaced by a screen showing the results of the last ICE execution (See Figure B-3). To return to "ICE Menu" screen, the user is prompted at the bottom of the screen (in yellow) to "press any key to cont-". If there exists more than one output screen display the user will need to press a key more than once, until the "ICE Menu" screen reappears.

There are no questionable links; however,
there are missing connections at the following pin locations:
Package T001 a 807400 is missing an output at pin 03

Press any key to continue

Figure B-3 ICE Execution Results

NONE: THE RESULTS DISPLAYED ARE THE OUTPUTS OF THE LAST EXECUTION OF ICE. THESE RESULTS ARE NOT CHANGED UNTIL ICE IS EXECUTED AGAIN.

Interpreting ICE results

The information provided by ICE can be categorized into two categories: missing connections and questionable connections. If the ICE program finds any missing connections, they are identified. If ICE finds any questionable connections, the end points of the link are specified.

Printing ICE Results

To get a print of the ICE results described above, the user must click the left button on the "Print" option of the "ICE Menu". The "Print" title will remain highlighted until the printing of results is finished. The user will be prompted with: "Ensure the printer is turned ON and then press any key to cont-". Once the user presses any key, the following prompt is displayed: "Printing is being initiated - press Return to cont-". Once the information has been sent to the printer, the "Printer" title will no longer be highlighted and the user may continue.

NOTE: THE USER MUST PRESS THE "RETURN" KEY TO ACTIVATE THE PRINTER DEFAULT OPTION, IF IT HAS NOT BEEN SET. IF IT HAS BEEN SET PREVIOUSLY, NO FURTHER ACTION IS REQUIRED. IF IS HAS NOT BEEN SET, THE USER WILL PROMPTED TO "PRESS ANY KEY TO CONTINUE".

Error Messages

1 - Temp.txt cannot be opened

Indication: The input file for ICE was not found. This file is created by the program INTICE.

Correction: a - Make room in the hard disk for the file.

b - If there is available space, increase the parameter FILES inside the CONFIG.SYS for 20. See the DOS manual for executing this task.

c - Verify the existence of the executable code INTICE.

2 - File ICLIST cannot be opened

Indication: The file ICLIST cannot be created, probably due to lack of space in the hard disk.

Correction: a - Make room in the hard disk for the file.

b - If there is available space, increase the parameter FILES inside the CONFIG.SYS for 20. See the DOS manual for executing this task.

3 - File Ymistake cannot be opened

Indication: The file Ymistake cannot be created, probably due to lack of space in the hard disk.

Correction: a - Make room in the hard disk for the file.

b - If there is available space, increase the parameter FILES inside the CONFIG.SYS for 20. See the DOS manual for executing this task.

4 - Routine CIRCLEV cannot open ICLIST

Indication: The routine CIRLEV tried to open the file ICLIST for reading and failed.

Correction: a - Make sure ICLIST is created and run the system again.

5 - Routine CIRCLEV cannot open ICCHIP

Indication: The file ICCHIP cannot be created, probably due to lack of space in the hard disk.

Correction: a - Make room in the hard disk for the file.

b - If there is available space, increase the parameter FILES inside the CONFIG.SYS for 20. See the DOS manual for executing this task.

6 - Routine CHIPLEV cannot open ICCHIP

Indication: The routine CIRLEV tried to open the file ICCHIP for reading and failed.

Correction: a - Make sure ICCHIP is created and run the system again.

7 - Routine CHIPLEV cannot open ICGATE

Indication: The file ICGATE cannot be created, probably due to lack of space in the hard disk.

Correction: a - Make room in the hard disk for the file.

b - If there is available space, increase the parameter FILES inside the CONFIG.SYS for 20. See the DOS manual for executing this task.

8 - Routine GATELEV cannot open ICGTMAST

Indication: The routine GATELEV tried to open the file ICGTMAST for reading and failed.

Correction: a - Make sure ICGTMAST is created and run the system again.

9 - File Rmistake cannot be opened

Indication: The file Rmistake cannot be created, probably due to lack of space in the hard disk.

Correction: a - Make room in the hard disk for the file.

b - If there is available space, increase the parameter FILES inside the CONFIG.SYS for 20. See the DOS manual for executing this task.

10 - If a power failure occurs or if for any reason the system is halted while executing ICE, chances are that the index of the TTLS and PINS are corrupted.

Correction: To recover the indexes run the utility batch file INDEXTTL.bat.

Bibliography

1. Adams, Capt Charles A., Jr. A Digital Circuit Design Environment. MS thesis, AFIT/GCS/ENG/87D-1. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987 (AD-A188831).
2. Deloria, Capt Wayne C. A Digital Logic Simulator with Concurrent Programming Considerations. MS thesis, AFIT/GCS/ENG/87D-10. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987 (AD-A188823).
3. Ehrhart, Capt Sue A. A Database Management System for Computer-Aided Digital Circuit Design. MS thesis, AFIT/GCS/ENG/88D-4. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1988.
4. Estes, A. and others. ICE - an Interconnect Expert, unpublished manuscript. Department of ENG, Air Force Institute of Technology, 1986.
5. Greenfield, Joseph D. Practical Digital Design Using IC's. New York, John Wiley & Sons, 1977.
6. Hayes-Roth, Frederick and others. Building Expert Systems. Massachussets, USA, 1983.
7. Kernighan, Brian W., Dennis M. Richie. The C Programming Language. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1978.
8. Mano, Morris M. Digital Logic and Computer Design. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1979.
9. Pressman, Roger S. Software Engineering A Practitioner's Approach. Second Edition, New York: McGraw-Hill Book Company, 1987.
10. Rich, Elaine. Artificial Intelligence. New York: McGraw-Hill Book Company, 1983.
11. Schildt, Herbert. Artificial Intelligence Using C. Berkeley, California: Osborne McGraw-Hill, 1987.
12. Schildt, Herbert. Advanced C. Second Edition, Berkeley, California: Osborne McGraw-Hill, 1988.
13. The TTL Data Book for Design Engineers. Second Edition, Texas Instruments, 1981.

14. Wagner, 1Lt Steven M. An Expert System for Discrete Component Digital Circuit Design. MS thesis, AFIT/GCS/ENG/87D-28. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987 (AD-A189680).
15. Waterman, Donald A. A Guide to Expert Systems. Reading, Massachussets: Addison-Wesley Publishing Company, 1986.
16. Winston, Patrick H., Horn, Berthold K. P. LISP. Second Edition, Reading, Massachussets: Addison-Wesley Publishing Company, 1984.

V I T A

Captain Jorge da Silva Santos was born January 01, 1949 in Rio de Janeiro, Rio de Janeiro, Brazil. He graduated from Escola de Oficiais Especialistas e de Infantaria de Guarda (School for Specialist and Infantry Officers), Parana, Brazil in 1976 with a Bachelor of Science (B.S.) degree and from ITA - Instituto Tecnologico de Aeronautica (Aeronautical Institute of Technology), Sao Paulo, Brazil in 1983 with a B.S degree in Computer Science.

As an officer in the Brazilian Air Force he was stationed at the Grupo Especial de Inspecao em Voo (Flight Inspection Special Group), where he was Chief of the Supply Section. He attended the Curso Superior de Tecnologia de Computacao (Superior Computation Technology Course) at ITA from February 1982 to December 1983. Upon graduation in December 1983, he was assigned to Centro de Computacao da Aeronautica de Brasilia (Aeronautical Computation Center of Brasilia) where he became Chief of Support Subdivision, Operational Systems Section, Systems Evaluations Section and Teleprocessing Section. In March 1987, Captain Santos entered the School of Engineering, Air Force Institute of Technology.

Permanent address: Estrada Intendente Magalhaes, 739

Vila Valqueire - Rio de Janeiro

Rio de Janeiro - Brazil - CEP 21330

END

6-89

DTIC